Fluid Simulation

Dhruv Kore, Giancarlo Gonzalez, and Jenny Sum
CS 488: Introduction to Computer Graphics
Professor Angus Forbes

1) Introduction

Figure 1.1: *Realistic water and ice simulation using Blender, courtesy of Andrew Price*

The occurrences of fluids are very common in everyday life, and are witnessed in many different forms outside of the realm of just liquids; for example, in smoke, fire, hurricanes, explosions, and other related phenomena. Fluid simulation has become a hot topic in computer graphics in recent years, due to its popularity and frequent use in special effects for feature films, acclaimed television series, and video games. Below, is an example of fluid simulation in the form of smoke, displayed in the feature film, Star Trek: Into Darkness, when the Enterprise uncontrollably spins, descending towards Earth.

Figure 1.2: *Scene from Star Trek: Into Darkness*

Fluid simulation has algorithms based in the field of computational fluid dynamics (CFD). In computational fluid dynamics, fluid, both in gaseous and liquid forms, are governed by partial differential equations that represent the conversation laws for mass, momentum, and energy. These partial differential equations are able to provide an accurate prediction of how fluid flows. Mathematical modeling, involving partial differential equations, numerical methods, involving discretization and solution techniques, as well as software tools, help aid scientists and engineers in performing experiments.

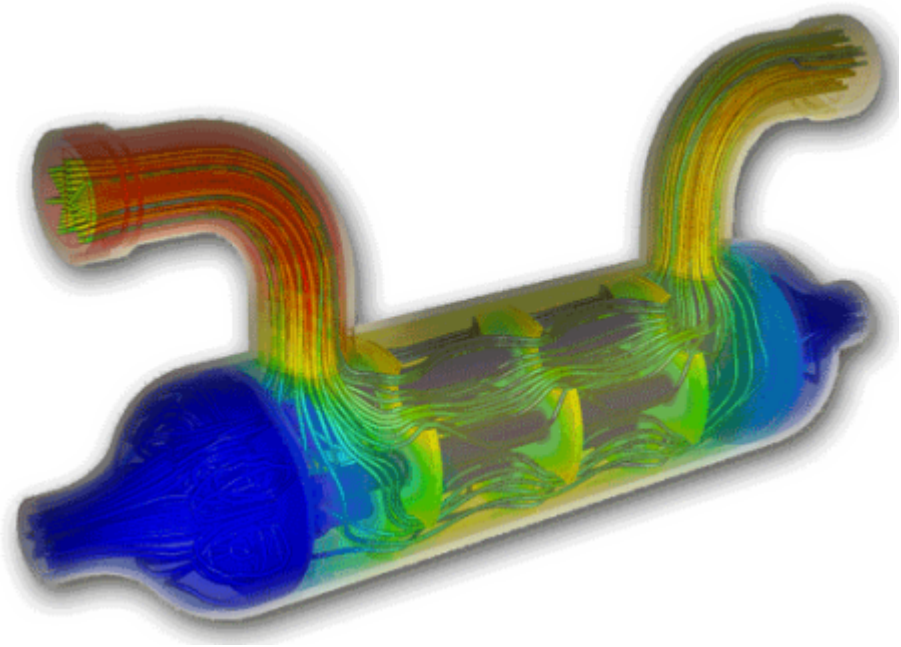

Figure 1.3: *Flow simulation through a pipe to foresee possible design issues, and optimize accordingly*

Computational fluid dynamics coupled by the help of fluid simulation in computer graphics help many professionals on a daily basis. For example, CFD permits architects to design comfortable and safe living conditions, aerospace engineers to improve aerodynamic characteristics, and meteorologists to predict the weather and gather warnings of natural disasters. Though CFD does not replace physical experiment measurements entirely, physical experiments can be expensive, slow, and sequential. CFD allows for a cheaper and faster alternative, as well as having multi-purpose uses.

In further sections, the two most general and widely used CFD equations that govern fluid simulation in computer graphics, the Euler and Navier-Stokes equations, will be discussed. There will also be a discussion on how fluid simulation has become what it is today, coming from very simple real-time particle systems, into extremely high quality animations that we have come accustomed to in our favorite movies, television shows, and games.

2) History of Fluid Simulation

Fluid simulation in computer graphics began with lower dimensional techniques such as the ad-hoc particle system. These lower quality techniques were used to create 2D shallow water models, and semi-random turbulent noise fields.

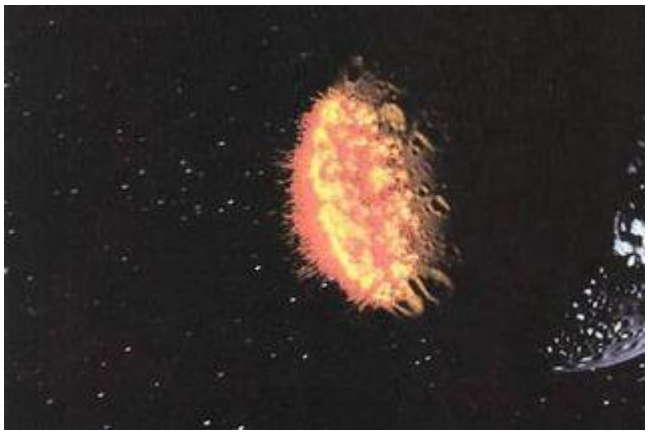

Figure 2.1: *Particle system used in explosion engulfing a planet in Star Trek II*

Particle systems were initially developed by William T. Reeves to create certain scenes for Star Trek II: The Wrath of Khan. He was trying to create an explosion effect that would engulf an entire planet. The use of particle systems is a technique used in which there is a collection of many particles that move together in such a way that it creates a fuzzy effect represented by simple shapes. These shapes are able to create different effects in fluid simulation. Below is an example of a particle system used to emulate fire. Though the fire is not very realistic, it is clear that the particles are simulating the movement of fire.

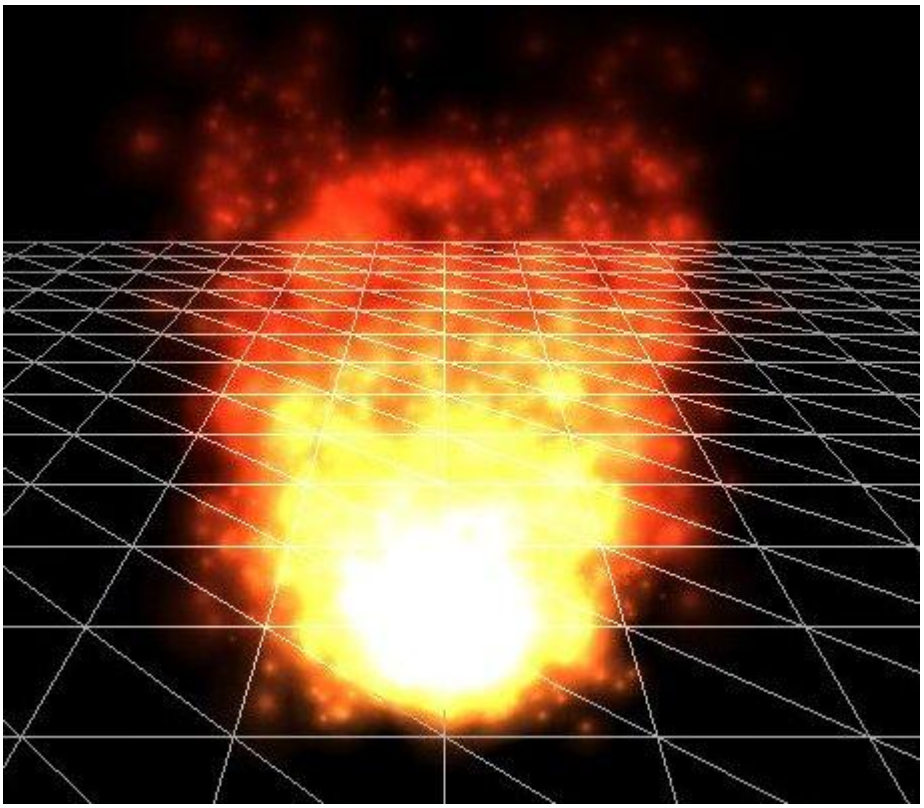

Figure 2.2: *Fire simulated by a particle system*

Shortly after the use of particle systems, Nick Foster and Dimitris Metaxas attempted to solve the Navier-Stokes equations in full 3D. This was the first time that a grid-based fluid simulation was used. Foster and Metaxas primarily worked off of a classic fluid dynamics paper that was written in 1965 by Harlow and Welch. In 1999, Jos Stam published a paper called Stable Fluids at SIGGRAPH, that used implicit integration of viscosity to provide stable behavior in fluid dynamics. This allowed for more realistic, and overall faster simulations. This technique allowed for Ron Fedkiw and others to develop more complex 3D water simulations thereafter.

3) Euler's Equations being used in a Grid/Particle system

One of the aspects of fluid simulation is viscosity. A paper titled "Particle-based Viscoelastic Fluid Simulation" attempts to create a particle-based simulation using viscosity. This simulation takes into account surface tension and tries to implement non-linear plastic behavior. This is achieved through the use of two techniques, the Eulerian grids and Lagrangian particles.

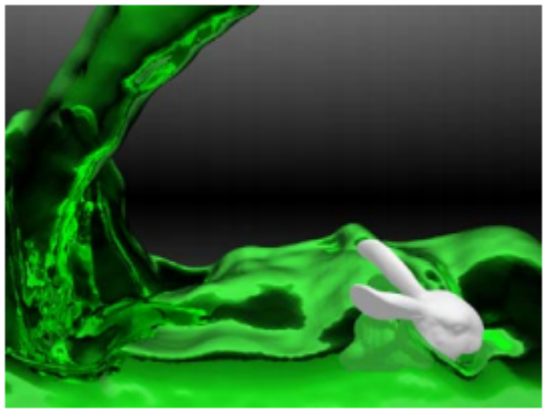

Figure 3.1: A heavy object submerged in simulated liquid

Eulerian grids use grid like behavior and control fluid flow within each section. The grids split the whole problem into grids that applies incompressible flow. Incompressible flow refers to the consistency of the density within materials that are submerged or surrounded by the simulated fluid shown in Figure 3.1. Since the change of density over time would imply that an object was compressed or expanded, this requires the divergence of fluid velocity to be equaled to zero.

The Eulerian equations depend on the conservation of mass, momentum, and energy. Eulerian grids do not conserve mass with each computation due to the fact that when a grid has a small feature, it may not recognize it. Combining the particle-based method, Lagrangian method, to the Eulerian grid method keeps track of material in clumps. Therefore, not losing any mass. The Lagrangian method uses a particle-based method with an implemented radially smoothing kernel so the particles are not visible. They are used just to keep track of mass.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

Figure 3.2: Eulerian equation for the conservation of mass

The Eulerian equation for the conservation of mass mostly depends on the density of the fluid, **ρ**, and the **u**, the fluid velocity vector. The Eulerian equation for the conservation mass is shown in Figure 3.2.

The momentum of a particle in this simulation depends on the surrounding particles and the difference between vectors and displacement of said particles.

$$\mathbf{D}_{ij} = \Delta t^2 P_i (1 - r_{ij}/h)\hat{\mathbf{r}}_{ij}$$

Figure 3.3: Density relaxation displacement equation that depends on the Pseudo-pressure of two particles

The density relaxation displacement equation, as shown in Figure 3.3, takes into consideration the pseudo-pressure of particles around the particle in question.

Algorithm 2: Double density relaxation.

```
1.  foreach particle i
2.      ρ ← 0
3.      ρ^near ← 0
4.      // compute density and near-density
5.      foreach particle j ∈ neighbors( i )
6.          q ← r_ij / h
7.          if q < 1
8.              ρ ← ρ + (1 − q)^2
9.              ρ^near ← ρ^near + (1 − q)^3
10.     // compute pressure and near-pressure
11.     P ← k(ρ − ρ_0)
12.     P^near ← k^near ρ^near
13.     dx ← 0
14.     foreach particle j ∈ neighbors( i )
15.         q ← r_ij / h
16.         if q < 1
17.             // apply displacements
18.             D ← Δt^2 (P(1 − q) + P^near (1 − q)^2) r̂_ij
19.             x_j ← x_j + D/2
20.             dx ← dx − D/2
21.     x_i ← x_i + dx
```

Figure 3.4: Pseudo code for Double density relaxation

Double Density relaxation is the process of finding two different densities of a particle. One of the densities depends on the number of neighbors and the other depends on the number of close neighbors.

Within the research paper mentioned above the code in Figure 3.4 is implemented. It takes each particle and computes the density and near density of each particle. The code also computes the pressure and near-pressure of each of the particles. After, it applies the displacements that were calculated.

Energy within fluids is the pressure within said fluid. The displacement due to pseudo-pressure, internal and external pressure, of the particle takes care of the conservation of energy.

Figure 3.1 demonstrates the use of incompressible flow and the use of Eulerian grids. Figure 3.5 in turn shows the effects of changing values on the object and how the simulated liquid reacts to it. Figure 3.6 shows what happens when the simulated liquid is "rained" onto an object. It drops liquid segments in random areas on top of the object.

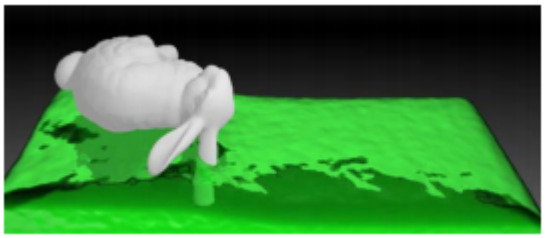

Figure 3.5: Liquid in an invisible tank with a floating object inside

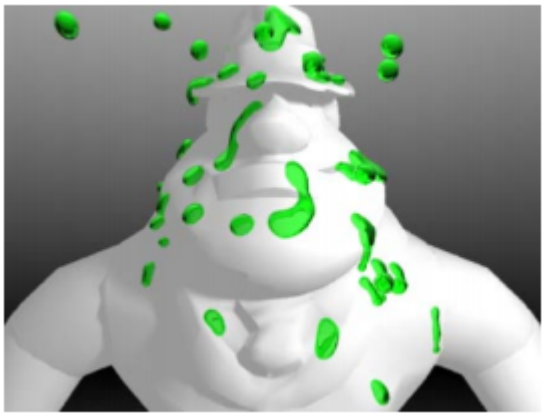

Figure 3.6: Demonstrates viscous rain on a character object

http://www.youtube.com/watch?v=1Q_zb65SXt0

Video 1: Demonstrations of Figure 3.1, 3.5, and 3.6

4) Navier-Stokes Equations and How They are Applied in Fluid Simulations

The Navier-Stokes Equations are a precise mathematical model for most fluid flows occurring in nature. Numerical algorithms weren't developed until about the 1950's and they are complex and time consuming because they strive for accuracy. In computer graphics, we sacrifice accuracy because we just

need to make them look convincing and be fast. Also, they are not as complex because they must be able to run on consoles that can run games, such as a smartphone.

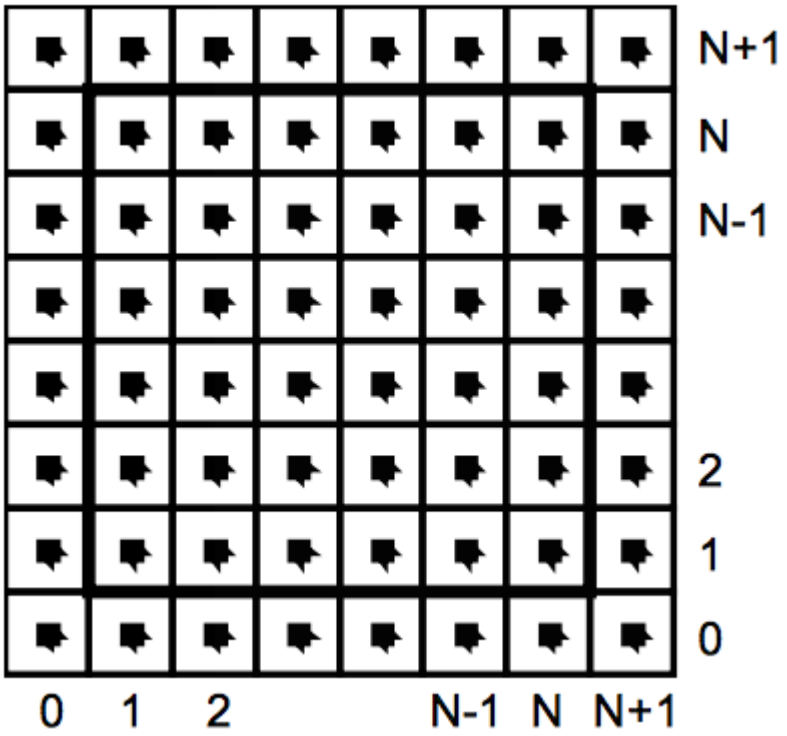

Figure 4.1: Computation grid. Both the velocity and density are defined at the cell centers. The grid contains an extra layer of cells to account for the boundary conditions.

In practice, we must use a finite representation of fluid. We accomplish a finite representation of fluids by dicing up the region of space into cells; at each cell we sample the fluid at the center. Fluids are modeled in a square grid as shown in Figure 4.1. In order to simplify the treatment of the boundaries, an additional layer of grid cells was allocated around the fluid's domain. In practice, the density and velocity of size = (N+2) * (N+2) are allocated in two arrays.

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

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla)\rho + \kappa\nabla^2\rho + S$$

Figure 4.2: The Navier-Stokes Equation for the velocity in compact vector notation (top) and the equation for a density moving through the velocity field (bottom).

In the density equation depicted at the bottom of Figure 4.2, it states that the change of density over time is due to three causes. Dust particles are simple to simulate because they are just carried along the velocity, as are any light objects. Something like smoke gets expensive to model. Since smoke has many particles we replace smoke particles with smoke density. A number between zero and one gives density: if zero, there are no smoke particles present; any other number gives us the amount of particles in that area. The first term states that density follows the velocity, second term states that it may diffuse at a certain rate, and the third term states that the density increases due to sources. These three ideas are used in solving density.

To resolve that density increases due to sources is simple. These sources are provided for a given frame in an array. The array is filled by a part of the game engine that determines the sources of density.

Diffusion is just a rate, and when it is greater than zero it spreads across the grid cells. A single cell's density decreases when lost to its neighbors, vice-versa it increases when gained from its neighbors. In order to have a stable method for diffusion an implementation of Gauss-Seidel relaxation is used. It is used because the densities, when changed, must go back to the ones initially started with.

The final step is to force the density to follow the velocity field. In order to have a simple and working implementation of advection two grids are used. One grid contains the previous density values, the second contains the new values. For each cell in the first grid, the cell's center position is traced

backwards through the velocity field. Then a linear interpolation is done on the previous grid cell and assigned to the current grid cell.

The Navier-Stokes equation for velocity field at the top of Figure 4.2 models a fluids state at any given instant of time. A velocity vector field assigns a velocity vector to every point in space. Imagine air occupying the room, if near a radiator the velocity of the air would be pointing upwards since heat rises. This is also evident when watching smoke rise from a cigarette or dust particle's movements. The Navier-Stokes equations describe the velocity field over time. If given the current velocity and forces, the equations give us the change of velocity over time.

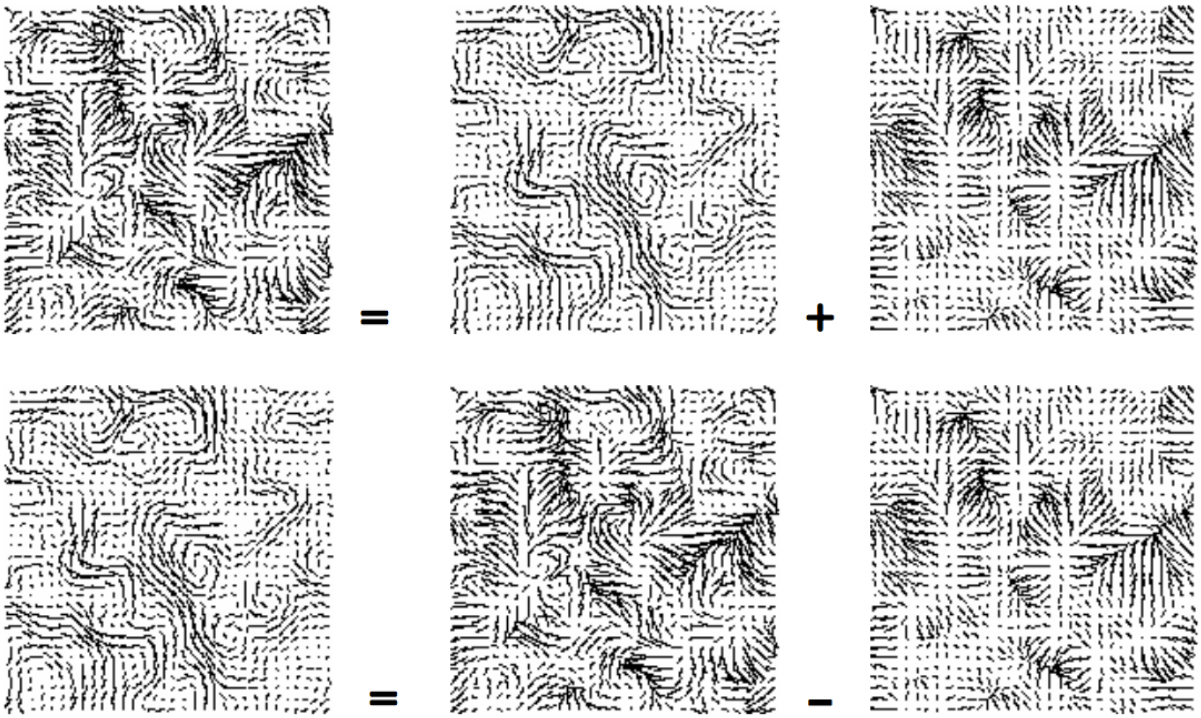

Figure 4.3: Every velocity field is the sum of an incompressible field and a gradient field (top). To obtain an incompressible field we simply subtract the gradient field from our current velocities (bottom).

In order to solve for velocity, velocity must be mass conserving and routine called project is used. This produces the swirly-like flows we see in fluids. Hodge decomposition states that every velocity field is the sum of a mass conserving field and a gradient field. This is illustrated in Figure 4.4 (top). The bottom of Figure 4.3 is the subtraction of the gradient field (a height field) from the current velocities. An incompressible field is now obtained.

A simple simulation of this is shown in the following link WebGL Fluid Experiment. It allows mouse movements to change the density sources and the velocity of the fluid. The higher the quality, the more particles involved. Resetting the particles puts them back into the initial state and stopping them will show them as a grid and not a moving fluid. With no density sources or velocity sources being updated a simulation cannot be performed.

5) References

Ash, Mike. "Mikeash.com: Fluid Simulation for Dummies." *Mikeash.com: Fluid Simulation for Dummies*. N.p., 3 Mar. 2013. Web. 13 Nov. 2014.

Clavet, Simon, Philippe Beaudoin, and Pierre Poulin. "Particle-based viscoelastic fluid simulation." *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, 2005.

Kuzmin, Dmitri. "Introduction to Computational Fluid Dynamics." *Institute of Applied Mathematics, University of Dortmund*.

Li, Xiaosheng. "Fluid Simulation for Computer Graphics." Institute of Software, Chinese Academy of Sciences. Web. <http://lcs.ios.ac.cn/intranet/images/2/2a/S-lxs.pdf>.

Price, Andrew. "Create a Realistic Water Simulation in Blender." *Blender Guru*. 9 Nov. 2011. Web. <http://www.blenderguru.com/tutorials/create-a-realistic-water-simulation/#.VGVrmVb4gzV>.

Shiffman, Daniel. "Particle Systems." *The Nature of Code*. Web. <http://natureofcode.com/book/chapter-4-particle-systems/>.

Smat, Jos. "Real-Time Fluid Dynamics for Games." *Real-Time Fluid Dynamics for Games* (n.d.): n. pag. *Research @ Alias Systems*. Alias | Wavefront, 1 Mar. 2003. Web. 10 Nov. 2014.

Stam, Jos. *Stable Fluids*. New York: Addison-Wesley, 1999. 121-128. Print.