CSE 328 Project - Realistic Fluid Animation

Bik Ghosh

SBID: 108776586

bikramaditya.ghoshdastidar@stonybrook.edu

**ABSTRACT**

In this paper, I attempt to implement a working version of a fluid simulator based on the paper *Realistic Animation of Liquids* published by authors Nick Foster and Dimitri Metaxas.

**Keywords**

Dynamic Fluid Simulations, Free-surface Flow, Navier-Stokes Equations, Physics-Based Modeling.

# INTRODUCTION

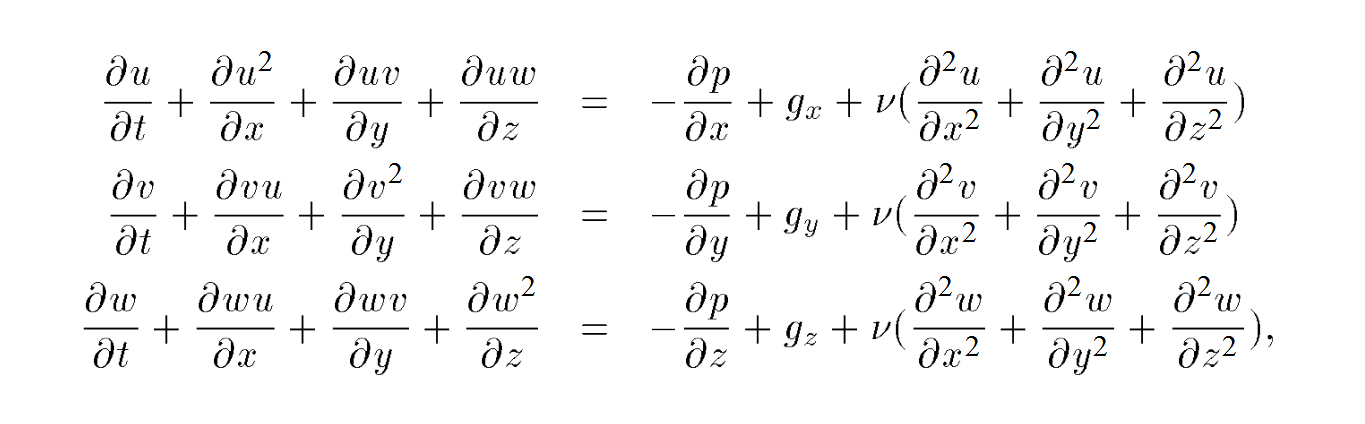
The paper Realistic Animation of Liquids models a way to animate fluid in a closed system using the Navier-Stokes equation as its basis. This project aims to solve the Navier-Stokes equation over a finite difference mesh approximation. Applied on a coarse grid, the Navier-Stokes equation is used to simulate a vector field complete with a profile of pressure and velocity per cell. This grid is then used to determine the position and behavior of the fluid surface. As this system is updated per frame, the motion of the fluid becomes apparent and animated through to completion.

Boundary conditions are specified for each cell that don’t behave like fluid. To keep the implementation simple, the grid is homogenized and to assume solid obstacles as fluid as well with special conditions.

We begin by describing the Navier-Stokes equations, which is discretized and then solved. We discuss how motion is determined in our grid-based system. We determine how pressure and velocities are determined in each cell, followed by how the free surface is controlled through the above approach.

# NAVIER-STOKES EQUATION

The Navier-Stokes equation for incompressible fluids is as follows:



where *u*, *v*, *w* are the velocities in the x, y, z directions respectively. Viscosity of the fluid is defined as *ʋ* and *p* is the pressure. The external body force, such as gravity, is defined as *g*.

The left-hand side of the equations account for the change in velocity in the fluid due to local acceleration and convection. The right-hand side of the equations takes account of external forces, the viscosity of the fluid, and the local pressure to determine the velocity change in the fluid’s subsystem.

With the above setup in mind, and assuming that not only momentum, but mass is also conserved, we attempt to discretize the equations and properly simulate fluid motion.

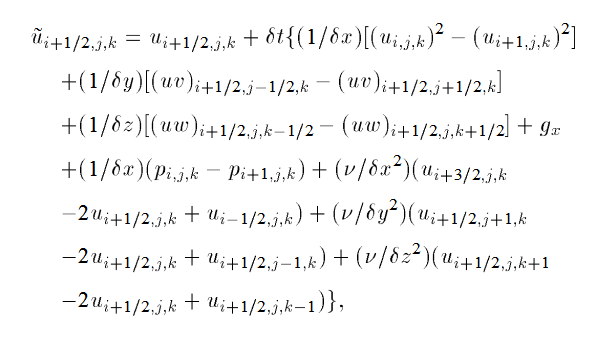
# APPLYING THE NAVIER-STOKES EQUATIONS

The implementation of this project has distinct steps.

* We realize that we want to discretize the otherwise continuous Navier-Stokes equation for our purposes.
* Then we need to determine our boundary conditions and apply it to our grid.
* We need a method to determine our free surface.
* We need to animate fluid motion.

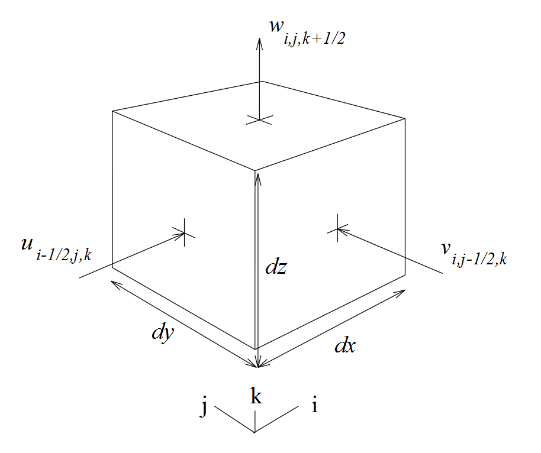
## Discretization

Our reference paper gives a discretized solution of the Navier-Stokes equation, and provides an equation for updating velocities for cells in our system:



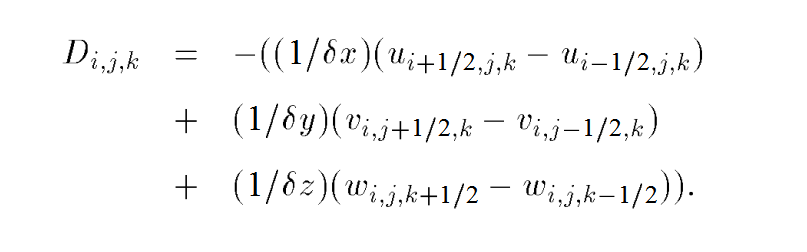
The above equation provides a discretized solution of the Navier-Stokes equation, which updates the *u* vector (in the *x* direction) for any cell (at *i, j, k* coordinates). This equation has been applied with the proper changes to update the *v* and *w* vectors for each cell.

The vectors for each cell are defined on the corresponding surfaces:



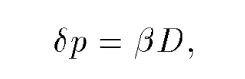
After the following iteration has been performed, we need to make sure that mass is conserved in the system and the fluid is incompressible. To do that, we have forced the divergence of each cell to be as close to 0 as possible.

For each cell at (*i*, *j*, *k*), divergence within the cell is given by:

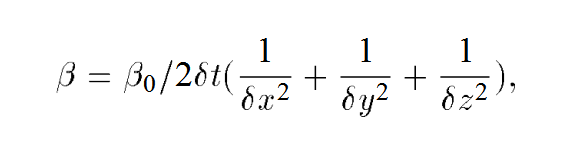


Since this is a discretized solution, a positive divergence means an influx of fluid, while a negative one means outflow. From this, we can update the pressure field in the system that will be used to update the velocity vectors.

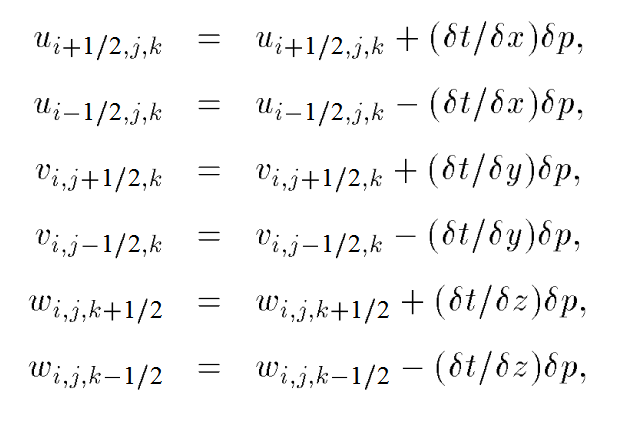
The pressure change in a cell is therefore:



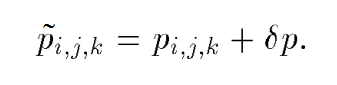
where *β* is given by:



Using this, we can now update the vectors of each cell to force incompressibility of fluid:



And we can update the pressure field as follows:



## Boundary Conditions

With the generalized solution to the Navier-Stokes now ready, boundary conditions must be set for cells that behave specially. Once these boundary conditions have been setup, the Navier-Stokes solution can be applied blindly to the entire system and we can be sure it will behave properly.

### Solid Obstacles

Boundary conditions must be set for solid obstacles, which obviously should not behave like fluid. In our system, the solid obstacles are implemented as *no-slip­* meaning they apply drag on adjacent cells. To ensure this, we set the solid cell’s velocity equal and opposite to the velocity that is tangent to an adjacent fluid cell. We also force the normal velocity to become 0, that keeps the liquid from intruding into solid obstacles.

### Inflow and Outflow

We must also set boundary conditions for inflow and outflow boundaries. In our system, we set a constant velocity for inflow locations, where fluid can flow freely into the system. For outflow locations, we let the velocities relax as normal, which lets the fluid exit the system freely. This ensures that the system can have good behavior with fluid entering and leaving the system.

### Free Surface

Boundary conditions must also be set for surface cells. Following the reference paper, the surface cells are individually updated using the solved Navier-Stokes equation, with the only difference that divergence is forcibly set to 0. This ensures that liquid stays compact yet incompressible throughout the runtime of the program.

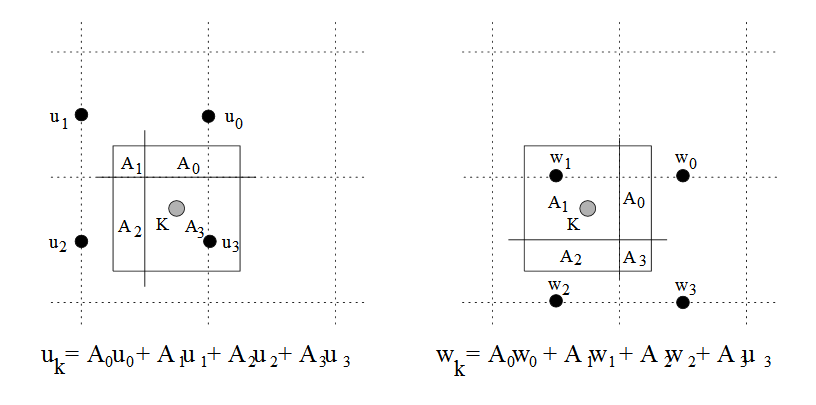
## Tracking the Free Surface

Now that we have defined a confined system that can update itself with the proper velocities and pressure, we now need a method to track the position and behavior of the free surface.

In this system, we are using massless marker particles to determine the position of the fluid surface, keeping the following in mind:

* A cell containing no particles is *Empty*.
* A cell containing any particles that is adjacent to an *Empty* cell is labeled as *Surface*.
* A cell containing any particles that is not a *Surface* cell, is labeled as *Full*.

Once the cells are labeled properly, we can convect these particles through our vector-pressure field, that will define the surface of the fluid in each frame – thus animating motion. As mentioned in the reference paper, an area-interpolation method is used to update the positions of the particles:



# IMPLEMENTATION

Now that we have defined everything necessary for our system to work, we move on to the implementation of the program.

## Technologies Used

The system is built as a Visual Studio 2015 project using the C++ programming language. Libraries are used to import OpenGL and other dependencies to ease the process of implementation. We are using the *nupengl.core* library to get everything necessary for an OpenGL project, such as *GL.h*, *glew.h* etc. We are also using the *SDL* library that helps us set up our application window and implement user interactions. We are also using the built-in OpenMP library for multithreading support.

## Project Structure

The project has two distinct components – the Core and the Application:

### Core

The Core contains all the classes and headers necessary to perform the computation that will construct and update a generic coarse grid following the Navier-Stokes solution that was previously discussed. The Core of this project is built in a reusable fashion that can be used in different ways based on the application’s requirements. Parameters and hooks are exposed that sets up the grid and conditions properly as required by the developer.

Models have been defined for *Grid*, *FluidCell*, and *Particles* that define the parameters and state for each component in our vector-pressure grid. An object-oriented approach has been taken to ensure that the implementation of the program is simple to follow and update. Each *Grid* contains a collection of *FluidCell*’s. Each *FluidCell* has corresponding *I, J, K* coordinates, that determine its position in the *Grid*. Likewise, each *Particle* has their corresponding *X, Y, Z* coordinates that are used to determine their position inside the *Grid*. With these models in our arsenal, we can now implement our program.

Most of the heavy lifting in the Core is done in the *NavierStokesSolver.cpp* source code file. The file contains all the logic necessary to apply the Navier-Stokes equations and the associated algorithms to our system. Each function in this file is implemented in a way that can really reap the benefits of a multi-CPU system, in order to increase performance. Since the particles and cells in any grid is updated individually, we can use the power of simultaneous computing provided by the OpenMP library.

### Application

The Application component uses the functionality exposed by the Core to open a window, set a static refresh rate, and prepare a grid for our simulation. In the provided source code, a window is opened with a 16x16 grid. Refresh rate is set to 60 frames per second. The grid is constructed with the lower cells surrounded by solid obstacles. The top left corner is marked as an Inflow boundary where new particles will be introduced at every time-step. The time-step for our simulation is set to *1/60* to create near-realistic animations.

# ALGORITHM

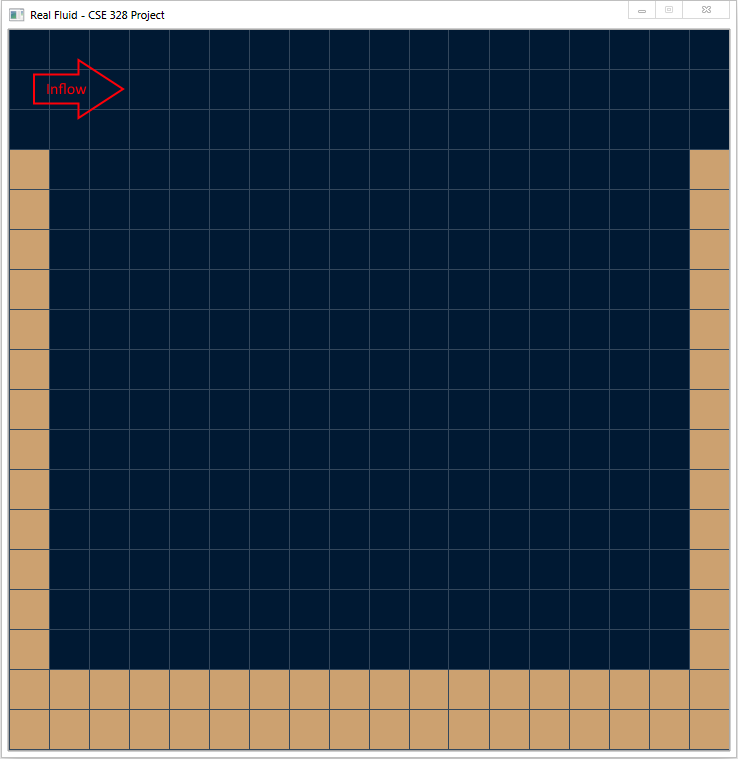
The summary of our implemented algorithm is as follows:

1. Define obstacles and starting fluid configuration, and place dynamic objects.
2. Set initial pressure and velocity conditions.
3. Determine cell contents depending on the method used to track the surface.
4. Set up boundary conditions for the free surface and obstacle cells.
5. Compute velocities for all *Full* cells.
6. Perform the pressure iteration for all *Full* cells.
7. Recalculate boundary velocities for *Surface* cells.
8. Update the position of the surface and objects.
9. Go to step 3

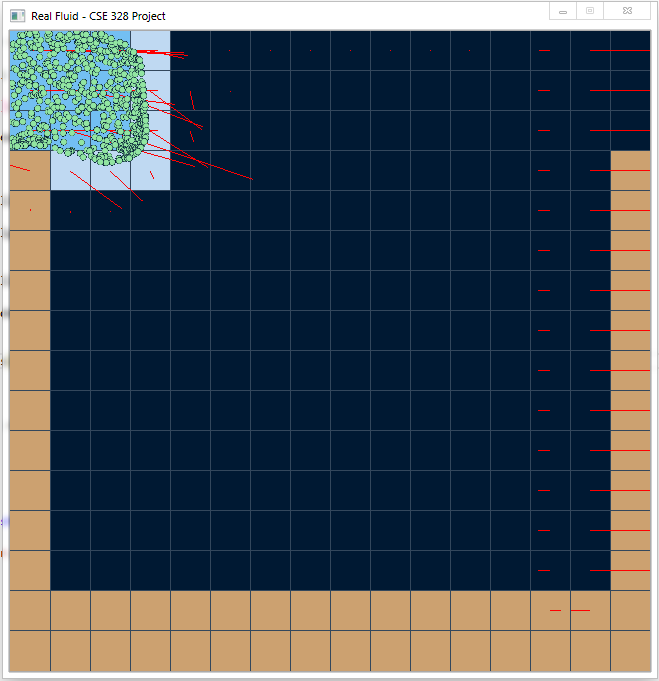
The above steps are reiterated every frame to animate fluid motion in our system. The particles’ positions determine the shape and position of the liquid. The free surface acts as the boundary of the fluid that define the proper structure of the liquid being animated.

# EXAMPLES

The system is set up as a 16x16 grid with the outer bounds defined as solid obstacles:



Once the simulation is started, fluid starts flowing into the system from the predefined Inflow boundary. Particles are continually introduced and they fall downwards due to gravity.



The green dots are the particles that defined the structure of the fluid. The blue-shaded cells are the corresponding *Full* and *Surface* cells. The red lines protruding from the center of the relevant cells are the vectors for those cells. The particles move according to these vectors, and follow sort of realistic properties of fluid.

# REFERENCES

1. Foster, N., and Metaxas, D. *Realistic Animation of Liquids* (Jun. 1993), <http://www.cbim.rutgers.edu/dmdocuments/gmip96%20Foster.pdf>.