A Fluid Implicit Particle (FLIP) Solver Built in Houdini

Alexis Agrotis

Master Thesis

MSc Computer Animation and Visual Effects

Bournemouth University, NCCA

August 2016

Abstract

The following thesis presents the research and implementation of a Fluid Implicit Particle (FLUID) . A literature review is included in order to highlight the most commonly used techniques for simulating fluids in the Visual Effects and Computer Graphics industries as well as critically analyse and compare them. A custom build solver was implemented in SideFX's Houdini using primarily gas microsolvers and VEX Wrangles or VOP networks. to simulate the different fields of the fluid as well as foam based on colour. The whole implementation process is broken down in detail justifying every step of the FLIP algorithm. Decisions were also made for the additional shading, lighting and rendering the simulation which required the transformation of particles into a surface. Various scenarios are described as well as possible future improvements that could be made.

Contents

Chapter 1

Introduction

Through the years of visual effects research and development, scientists and artists in the industry have tried to control and art-direct fluid simulations to its maximum. Fundamental aspects of fluid simulations are interactions between the particles, as well as interactions between the main fluid body and static objects serving as obstacles in the fluids' way of expansion[Lundberg, 2012]. Perfecting these aspects, as well as simulating the alteration regarding the volume of the fluid, can be computationally heavy and complicated[Thürey et al., 2007]to simulate, but eventually they give realistic fluid flow in a scene. Addressing the fluid incompressibility, researchers have managed to achieve convincing realism by assuming that the fluid's volume is preserved during the simulation[Ghourab, 2011]following complex mathematical equations and laws of physics. By solving those equations along with the introduction of custom velocity fields applied into fluid solvers, they managed to gain more control over the behaviour of the fluid.

According to Lundberg [2012], "a fluid solver is a model that solves a set of equations for describing a fluid phenomenon as smoke, fire and liquids". Depending on the desired final outcome and the nature of the fluid, the solver properties are adjusted accordingly. These solvers already exist in current mainstream 3D packages, like Houdini, however they lack in providing the artist the ability to art direct the fluid motion to the fullest since they are not extendable or they are hard to be internally modified by users without technical backgrounds and mathematical knowledge [Claes, 2009].

Combining together different components from multiple solvers is an effective tactic followed over the years with one solver addressing disadvantages and problems of the other in different occasions and vice versa [Cornelis et al., 2014]. Houdini provides the so called "microsolvers" which can be built in a node fashion setup inside the Dynamic Operators (DOPs) Network, and in combination with Wrangle nodes (nodes that allow the developer to write VEX code) or VEX Operations (VOPs) Networks (graphical representation of VEX code) solve the mathematical equations that are vital for fluid simulations [Claes, 2009].

In this project, an extensive use of microsolvers is proposed, in order to

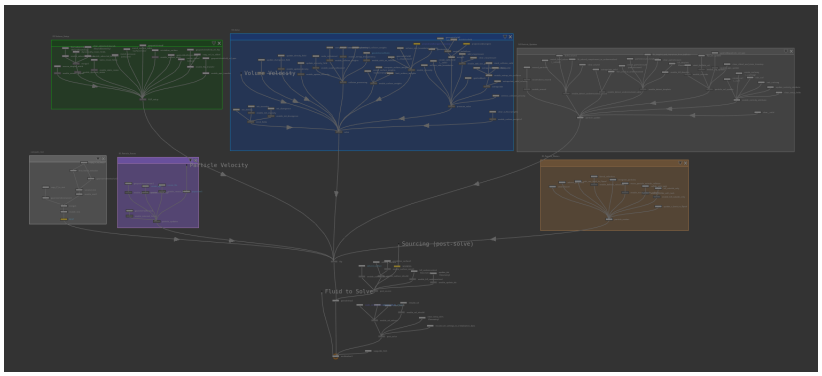

Figure 1.1: The complicated network of the built-in FLIP solver in Houdini

simulate the behaviour of the water body in multiple scenarios. As it will be discussed later, colour is the fundamental grouping factor of the fluid's contained particles, as well as the driving element for the acceleration changes in velocity and position and creation of foam.

This thesis is structured in five chapters. Chapter 2 presents the literature review of previous work regarding fluid simulations; Chapter 3 presents and compares the technical background and mathematical approaches that solvers use to simulate fluids; Chapter 4 discusses the in-depth implementation process for building a custom solver in Houdini's DOP network; Chapter 5 concludes the paper and make suggestions on how the whole project could be improved.

Chapter 2

Previous Work

The Fluid Implicit Particle (FLIP) method was introduced as an improved and simplified model to the previously published approach of Particle In Cell (PIC) [Brackbill et al., 1988]. PIC (analysed in detail in Chapter 3), represents the earliest hybrid approach where a grid evaluates the particles that represent the fluid's motion and properties**[Harlow, 1963]**. The reason why Bridson et al improved Harlow's PIC and proposed the FLIP method was due to the fact that PIC main issue was the immoderate numerical dissipation resulting from calculating the mean value of the particle's weight into the grid cell and then interpolating them back onto the grids [Ghourab, 2011]. Consequently, the outcomes of the simulation had a relatively more uniform and smooth representation of the fluid in the simulation.

Researchers have not limited the use of FLIP only for simulating fluids. Zhu and Bridson [2005] proposed a physics-based approach for animating sand, avoiding studying the granular nature of sand and considering it as a sand continuum. They managed surface advection by using a cloud of particles and proved that a fluid simulator can be turned into a sand simulator following modifications and additions describing the sand behaviour. The boundary conditions and the incompressibility were also addressed in this paper using an "auxiliary grid". Other application of the FLIP method was introduced in Calvimontes [2013] for simulating "spreading, wetting and penetration of fabric" and in Lentine et al. [2012] for experimenting with larger time steps. Specifically they used a colour function that would add "forwardly advected semi-Lagrangian rays" and represent the portion of fluid inside a region in order to preserve lost mass conservation, handling water as a free surface. The latter, means that the velocities should be extrapolated "across the interface which can create artefacts at large time steps".

In a similar fashion to Zhu and Bridson [2005],de Goes et al. [2015] departed from Lagrangian methods and considered "fluid particles no longer purely as material points, but also as volumetric non-overlapping fluid parcels that partition the fluid domain" in an attempt to combine strong incompressibility and low numerical dissipation something that Lagrangian approaches lack. They

managed to find a way around and avoid large amounts of damping caused by kernel-based evaluations of the internal forces and added an auxiliary grid that gave "more accurate and efficient pressure projections", energy dissipation and velocities of particles transferred back to the grid.

Moreover, Cornelis et al. [2014] introduced the hybrid approach of Implicit Incompressible Smoothed Particle Hydrodynamics (IISPH) for pressure projection on a Lagrangian grid, combined with FLIP (with an additional degree of freedom for the particles) for boundary enforcement in order to simulate incompressible fluids. They studied in depth and critically analysed and compared their IISPH-FLIP (which in simpler words is "an extension of SPH with FLIP particles") approach with pure SPH and FLIP methods, and succeeded on the accurate preservation of mass and volume of the fluid, a common problem for FLIP simulations solved on Eulerian grids. Furthermore, regarding the benefits against pure SPH, their method had proved to be more efficient towards computing detail and reduction of numerical diffusion, but higher compared to FLIP resulting to a smoothed velocity field with less detail. It is a proof of what it has been said earlier that by using hybrid approaches one approach can cancel out the drawbacks of the other and vice versa.

Regarding specifically the built in Houdini solvers, many studies have tried to reconstruct alternatives to those from scratch allowing more manipulation of the effects by custom setups. Ghourab [2011] implemented an alternative solver to the Pyro tool in Houdini focusing notably on performance issues, namely faster execution of the simulation which would be in need of less secondary storage. Furthermore, Claes [2009] followed a series of experiments that resulted into modifying already existing solvers, not setting them up from zero, and which they would be able to be used in a variety of applications. They managed in that way to prove that the introduction of custom fields may allow broader uses in a number of different simulation scenarios.

Extending the idea of setting up solvers that would include additional functionality compared to Houdini's solvers, Lundberg [2012] focused on art directing "secondary water effects" namely "sprays, foam, splashes and mist" that would be implemented into the simulation. The main body of the fluid followed the FLIP approach, while secondary effects were based on particle systems in fluid dynamics as well as the handling of surface collision and custom velocity fields.

Chapter 3

Technical Background

In this Chapter an analysis of the algorithms and equations that need to be followed in order to simulate fluids are presented.

3.1 Navier-Stokes equations

The incompressible Navier-Stokes equations are a group of partial differential equations that have been used over the years in order to simulate the motion of fluids (liquids and gases) serving as the base of most solvers [Ghourab, 2011] in Computational Fluid Dynamics (CFD)[Calvimontes, 2013]. The result answers the question of "what factors affect the flow of a fluid" Wrenninge [2011] by returning a vector field describing the change of velocities [Lundberg, 2012]. According to 9, this differs from Rigid Body Dynamics (RBD) where the results describe the course of the particles' position. The following breakdown and explanation of the general formula follows the milestone book of Robert Bridson "Fluid Simulation for Computer Graphics" Bridson [2008].

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

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

where:
u the velocity of the fluid
ρ the density of the fluid (specifically for water: 100 kg/m3)
p the pressure
g the external forces
ν the kinematic viscosity, i.e. how viscous the fluid is

The general formula is consisted of two parts: the momentum equation [Ghourab, 2011] and the incompressibility condition [Claes, 2009]. These two parts include the terms of advection, diffusion, external forces and pressure/incompressibility which will be explained while breaking down the Navier-Stokes equations.

(2)Advection describes how moving the fluid influences its inner variables. A semi-Lagrangian approach was adopted by [Stam, 1999] in order to solve the advection part. This approach is considered stable since it guarantees that "the velocity computed cannot be higher than the velocities in the field"[Salomonsson, 2011] which is an outcome for using bilinear and trilinear interpolation.

Diffusion lets velocities to move from their current point to a new one and along with the viscosity amount controlling the speed of which this will occur.

External forces is the sum of all external forces represented by g on the equations such as gravity, wind and temperature.

3.1.1 Momentum Equation

The momentum equation takes into account the forces reacting on the fluid and then describes the way the fluid accelerates. The equation derives from Newton's second law where

$$\vec{F} = m\vec{a} \tag{3.3}$$

and the acceleration of each particle can be denoted as

$$\vec{a} = \frac{D\vec{u}}{Dt} \tag{3.4}$$

Gravity is the main force influencing the motion of fluids with forces and by definition

$$m\vec{g} = (0, -9.81, 0)m/s^2 \tag{3.5}$$

The relations in between particles can be considered when calculating pressure. Pressure is in charge of moving the fluid from areas with high pressure to areas with lower pressure based on its calculated gradient field . In the cases where there is equality of pressure towards all the directions, the pressure becomes zero and consequently no acceleration is achieved. Additionally to gravity and pressure, viscosity is another force influencing the flow which pushes the fluid towards the average velocity of nearby particles. . In the case of simulating water the viscosity factor is most of the times equal to zero and in that way the Navier-Stokes equations become slightly simpler to solve since a component is taken out [Salomonsson, 2011]. Without this component, the fluid would be named "inviscid" and transform the Navier-Stokes equations into "Euler equations"

$$\frac{D\vec{u}}{Dt} + \frac{1}{\rho}\nabla\rho = \vec{g} \tag{3.6}$$

$$\nabla \cdot \vec{u} = 0 \tag{3.7}$$

3.1.2 Incompressibility equation

The incompressibility equation (2) ensures a non-divergence (or divergence free) velocity field (set always to zero) that preserves the mass of the fluid throughout the simulation [Claes, 2009], i.e. making it incompressible and also enforcing boundary conditions on it [Salomonsson, 2011]. Regarding the latter, i.e. the case where the fluid collides with a solid object, the normal component of velocity needs to be equal to zero resulting to

$$\vec{u} \cdot \hat{n} = 0 \tag{3.8}$$

with being the normal to the boundary. Different cases such as animated objects are not examined in the project's implementation thus they are not presented in detail here.

Going back to the incompressibility condition and according to Bridson, if we think an area of the fluid in time and call the volume and its boundary surface the measurement in volume change can be calculated "by integrating the normal component of its velocity around the boundary"Bridson [2008] .

$$\frac{d}{dt} volume(\Omega) = \iint_{\partial\Omega} u \vec{\cdot} \hat{n} \tag{3.9}$$

So consequently in our case, where we want to ensure the incompressibility condition of the fluid, the volume would be equal to zero and by using the divergence theorem to turn the equation into a volume integral we will eventually get for only Ω

$$\iiint_{\Omega} \nabla \cdot \vec{u} = 0 \tag{3.10}$$

and for the whole body

$$\nabla \cdot u \vec{=} 0 \tag{3.11}$$

Pressure is making sure that the fluid stays divergence-free throughout the simulation. Since pressure is present only in the momentum equation, Bridson

integrated it to the divergence of the velocity by introducing the equation for pressure

$$\nabla \cdot \frac{1}{\rho} \nabla p = \nabla \cdot (-\vec{u} \cdot \nabla \vec{u} + \vec{g} + \nu \nabla \cdot \nabla \vec{u}) \tag{3.12}$$

There are two main approaches of solving Navier-Stokes equations based on the way attributes are stored [Fowler, 2013], the fluid representation, and the desired outcomes. The first is a grid-based approach (Eulerian) and the second is a particle-based approach (Lagrangian). Additionally, hybrid approaches have been introduced where the fluid is described as particles flowing through a grid [Ghourab, 2011], with FLIP being the most important example, as well as the Lattice Boltzmann[Calvimontes, 2013] and Vorticity-based methods [Ioannidis, 2012]. According to Calvimontes [2013], Lattice Boltzmann methods (LBM) are relatively new approaches that are based on "nanoscopic models and microscopic kinetic equations" in contrast with the rest of the methods that solve the equations of macroscopic properties, like mass, momentum and energy. It is a simple approach to understand and therefore implement, however they lack scalability and they have minimized time steps. No further detail will be presented for this method since the focus of the project is different.

Lentine et al. [2012], solved the equations by calculating initially an intermediate velocity field u* using the standard advection process of a semi-Lagrangian approach which utilizes the approximation of the velocities of the area where the fluid's body is travelling across. In the remaining area, namely where the fluid is not passing by, the velocities are obtained by the "closest-point velocity extrapolation" process close to the surface.

3.2 Lagrangian approach

In the Lagrangian approach, the fluid variables are stored on particles which represent the body of the fluid and they are uniquely identified by their position and velocity . Moreover, while they are moving they transfer information like acceleration, density, pressure and viscosity [Ioannidis, 2012] which are taken into account while the equations are solved. These particles have no boundaries surrounding them (unless explicitly defined for apparent collision detection situations) resulting on covering an infinite space. Particle-based solvers tend to be useful when the visualisation of splashes is desired [Fowler, 2013].

A typical example of the particle based method is the Smoothed Particle Hydrodynamics (SPH) and it is mostly used for calculating the interaction forces of the particles[Zhu and Bridson, 2005] "derived from smooth kernel functions" [de Goes et al., 2015]. Whenever the velocities are updated, the projection of the pressure is undertaken on each particle and in the case the particles are relatively close to each other an "explosive behaviour tends to rise" [Lundberg, 2012] and can be avoided by the increase of the substeps during the simulation execution.

3.3 Eulerian approach

In the Eulerian approach (Stan's Stable Fluids), values of velocity are stored on 3D grid, fixed points sampled with a fixed length $\Delta_x, \Delta_y and \Delta_z$between the grid points and calculate the changes of the quantity q over time[Salomonsson, 2011]. The grid represents the space's finite volume within which the fluid flows. This volume data or fields can be of two data structures; scalar field and vector gradient field. The former is a Signed Distance Field (SDF) serving as the distance from a point to the surface and keeps float values for the cells on the grid and represents density, viscosity and temperature. The SDF returns three values after comparing the distance and the point; positive for points inside the object, negative for outside and zero for voxels on the surface. The latter serves as the evaluation of the direction towards the areas of change that takes place in the scalar field. Instead of storing a float value, it stores a vector of floats for each cell on the grid and can serve as the representation of velocity that is advected by the pressure projection included in the Navier-Stokes differential equations, forces or colour (RBG values).

This grid acts as a boundary enforcement body that limits the fluid's body to move further from it and only move at a specific area. Grid-based methods store the velocity, pressure, the position of fluid and "any additional variables on a fixed grid" [Zhu and Bridson, 2005]. Moreover, by being combined with a "level set based surface tracking algorithm" [Calvimontes, 2013], they can come up with a very accurate fluid visualisation of behaviour and they are mostly used when simulating liquid, smoke or fire [Fowler, 2013].

According to Ghourab [2011], the Eulerian approach is the most commonly used one within software packages off the shelf. In order to improve the flow of the fluid, it is possible to add custom fields of forces, such as wind or gravity that will ultimately be combined and added to the solver's structure as a single force, what is included in the Navier-Stokes equations as g.

3.3.1 Different kinds of grids

In order to depart from the main issue of numerical dissipation in the Eulerian method, volume data structures have been proposed through the years mostly towards redefining the grid behaviour.

The most commonly used structure for level set, scalar and particle fields is the Collocated Grid where the variables are kept at the centre of each grid cell. According to Salomonsson [2011], each cell represents the area in between a "sample point and its positive neighbours".

A second broadly used grid structure, proposed by **Harlow [1963]** is the Staggered Grid which is also known as Marker and Cell (MAC) grid. Its main difference with the collocated grid is that the variables are stored at different positions, not necessarily the centre. The pressure is sampled at the centre and the velocity at the "Cartesian counterpart" [Salomonsson, 2011] which indicates the split point between the connected lines of the grid. Interpolation is valuable for acquiring the value of the velocity at a given position on the grid, whereas

bi- or trilinear interpolation for each component is used for getting the velocity at a random point and return a vector value. Bridson 2007 states that in this fashion the amount of fluid moving along the grid cells can be calculated.

Museth [2013]in Digital Domain has designed the DB-Grid which when compared to the open source volume data structure Field3D by Sony has been proven to be slightly faster. Its main objective is to deal with sparse volume data so when the fluid body is acquired onto the grid only the cells containing the fluid are stored instead of storing the whole grid. According to **Johansson**, this gives a good representation for Signed Distance Functions (SDF) which will discretize the vital information about the fluid body from the empty cells. Finally, DB-Grid offers both fast random and sequential access accordingly to the computation time been constant or linear respectively [Johansson, 2010].

The adaptive grid, as mentioned by Brackbill et al. [1988], is "generated by solving a variation problem in which one minimizes a functional". The functional may be manipulating the smoothness or spacing for example. The adaptive grid can be useful when used in PIC simulations; however there might be limitations with regards to low-speed flows that vary in pressure and densities.

3.4 Comparison

Depending on the objectives of the simulation to be implemented and the desirable outcome to be visualised, it is crucial to critically compare the existing and proposed algorithms regarding their performance.

3.4.1 Advantages

The Lagrangian approach tends to minimize storage cost and provides the ability of tracing accurate points reasonably [Salomonsson, 2011]. This is due to its complexity reduction since particles have no fixed connectivity between each other and large motions are handled in a simpler way (14). Additionally, this approach is well known for the mass conservation ability as well as the fields of pressure and density calculated from the "weighted contributions of neighbouring particles rather than solving linear systems of equations"[Ioannidis, 2012] .

The Eulerian approach tends to be simpler towards tackling the problem of incompressibility conditions of the fluid which is one of the most important factors when implementing a solver [Zhu and Bridson, 2005]. The use of a grid provides less complicated numerical approximations since the points in need for evaluation are fixed and not unpredictable in space. Consequently, the method provides more accurate visual results with "smooth liquid surfaces and large time steps" [Calvimontes, 2013] by preserving the fluids surface representation in disregard of the number of particles in the system.

3.4.2 Disadvantages

Regarding the Lagrangian method, numerical artefacts may cause major problems on the vividness of the fluid motion [Ioannidis, 2012]. Even though the non-fixed connectivity is considered as an advantage, following the links between particles can be very hard. Furthermore, using a large number to simulate the fluid motion can cause memory problems when finer detail is desired to be produced .

Even though the Eulerian approach provides realistic and simple outcomes, it has performance restrictions regarding speed and memory, as well as the overall control, scalability and resolution of the final result. However, compared to the Lagrangian method it is proved to be faster since there is no "interpolation of the fluid to and from the particles process"[Ghourab, 2011]. Concerning the memory, Eulerian simulations tend to be heavier in computational time since empty cells (no fluid included) of the grid are stored as well and the resolution is mainly conditional to the size of the grid [Ioannidis, 2012]. Furthermore, according to Zhu and Bridson [2005], they struggle calculating the advection part of the Navier-Stokes equations causing large amounts of numerical dissipation (13) due to a group of interpolation errors introduced by the semi-Lagrangian method[Stam, 1999]. In this method, the average value of the previous time step is calculated, thus at each advection step the averaging is been calculated. Averaging these values result into a smoother surface which consequently reduces the amount of detail[Bridson, 2008]. Finally, Fowler [2013] states that in some scenarios particle-based simulators may need a large amount of particles to visually achieve high resolution and quality.

3.5 Hybrid

In scenarios where either Lagrangian or Eulerian approaches did not satisfy or meet the criteria for the desirable effects simulation, developers have undertaken hybrid approaches. In this case, the fluid is represented by particles that flow through an Eulerian grid. According to 1, the advection is controlled by particles and the rest of the "fluid quantities are integrated on the grid". Hybrid methods where mainly developed in order to minimize issues on advection and pressure for relations between particles[Bridson, 2008]. These issues once again regarded incompressibility, since pressure projection seemed to be coupling together the velocities in order to ensure having a divergence-free velocity field, while the velocities where stored on particles and were been advected on space.

3.5.1 Particle In Cell (PIC)

As it is mention in the introduction, the Particle In Cell (PIC) method dates back to 1963 where Harlow [1963] introduced this hybrid idea for solving compressible flow where the advection step was handled by the use of particles and the rest of the simulation steps - diffusion, external forces, pressure projection, boundary conditions - been computed on a grid.

The main algorithm for PIC starts with all the variables of the fluid already kept on particles and are representing the fluid body. It is a common practise to seed eight particles in each cell of the grid and at each time step iteration the velocity is transferred from the particles to the grid. Eventually, the grid interpolates back to the particles which are then advected in the grid velocity field. A normalized weight W regarding velocity of nearby particles is calculated for every grid point for more accurate and robust particle to grid transfer.

One of PIC's main drawbacks is that it suffers for major numerical dissipation due to the weighted average and interpolation of velocities for transferring. This results into smoothness or sharpness loss, that would be interpolated to the particles from the grid and take away any choreography control over fluid simulation for artists. In simpler words, any particle characteristic is lost due to the velocity been overwritten presenting a more uniform fluid flow.

The kernel function k, as proposed by Bridson, should be added to the grid spacing which will determine whether the particles will disappear from the grid and not contribute to the grid given Δ_x is less than the support. In the cases where Δ_x is larger, the function will result into the previously mentioned "smoothness" that is an undesirable outcome. The kernel function could possibly be a Gaussian, a spline

$$k(s) = \begin{cases} (1-s^2)^3 : & s < 1 \\ 0 : & s \geq 1 \end{cases} \tag{3.13}$$

or a trilinear hat function

$$k(x,y,z) = h(\frac{x}{\Delta_x})h(\frac{y}{\Delta_x})h(\frac{z}{\Delta_x}) \tag{3.14}$$

with

$$h(r) = \begin{cases} 1-r : & 0 \leq r \leq 1 \\ 1+r : & -1 \leq r \leq 0 \\ 0 : & otherwise \end{cases} \tag{3.15}$$

The full-particle PIC described by Brackbill et al. [1988] eliminates numerical diffusion by using a full-Lagrangian representation of the fluid where each particle caries all the properties of the fluid as well as the momentum and energy. This approach differs from the classical PIC method in whether or not the particle variables are preserved at each time step. While the former describes the convection step only and particle data are replaced by grid solutions, the full-particle PIC describes the whole simulation cycle and particle data are updated, not overwritten.

3.5.2 Fluid Implicit Particle (FLIP)

Having said that the main issue of the PIC method was numerical dissipation, the Fluid Implicit Particle (FLIP) was introduced in order to solve this problem.

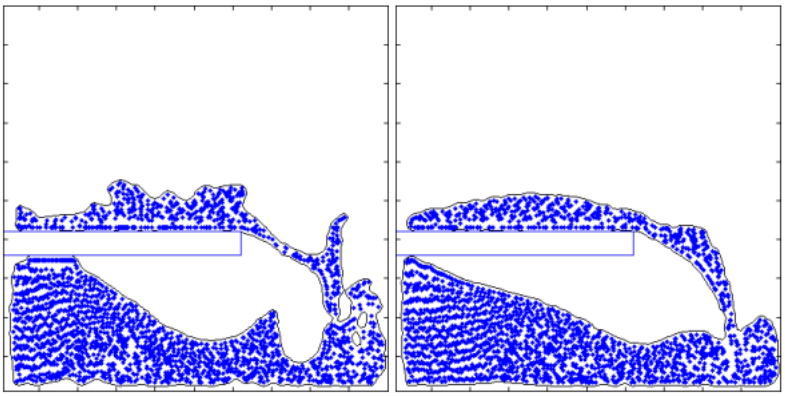

Figure 3.1: Comparison of FLIP and PIC velocity updates[Zhu and Bridson, 2005]

According to Zhu and Bridson [2005], FLIP is preferable over PIC for simulating inviscid flows like water, while PIC is better for viscous flows, like honey, due to its smoothing nature. The particle system takes over the representation of the fluid volume and along with it, the auxiliary grid is used for incrementing the particles based on the changes calculated on the grid. Additionally to reducing the numerical dissipation to minimum, Brackbill and Ruppel also achieved variations in the data, which visually gave a more vivid fluid flow in the simulations' results over the years.

The algorithm of FLIP defers from PIC by using the change of quantity calculated on the grid to increment the particle value, not overwrite it. For each of these increments only one smoothing, that is not gathered, takes place from interpolation of the grid. FLIP breakdown representation could be written as

- Transferring of particle values to the grid
- Extrapolation on the grid and storing of the grid values
- The rest of fluid variables, such as pressure, are calculated on the grid
- Return the updated grid value
- For each particle the change will be interpolated by subtracting the old grid value from the new
- The difference from this subtraction is added to the particle's value
- Advection of the particles in the grid velocity field

A known drawback when using FLIP, and it will be discussed in the implementation chapter as well since it was occurred during testing of the solver, is the development of noise making it unstable. This issue arises since the particles are allowed to move more freely than when in the grid, and sometimes might result less accuracy. Salomonsson [2011] recommends the blending of a small degree of the PIC algorithm, that would clearly not add any numerical dissipation, but help this explosive behaviour be tamed. The algorithm would then be modified, as recommended by Zhu and Bridson [2005], to the new one regarding incompressible flow:

- The particle velocities and positions need to be initialized and then for each time step
  - Calculate the weighted average of nearby particle velocities for each cell on the grid. Nearby particles are inside a cube twice the width of the cell and positioned on the centre of the grid point
  - For FLIP: keep the grid velocities
  - On the grid: carry out the non-advection steps of a standard fluid simulation, such as adding the acceleration of gravity to the grid velocities
  - For FLIP: get the interpolated difference between the new grid velocities (by projection step) and the stored ones and add them to each particle
  - For PIC: the new grid velocity have to be interpolated to the particles
  - Particles need to be moved through the grid velocity field and outside of solid wall boundaries
  - Return the particle positions that would update the fluid motion

The equation of the particle acceleration been interpolated from the grid data as presented in the third step of the algorithm and also proposed by 8 is

$$\frac{du_p}{dt} = \sum_v \frac{dU_\nu}{dt} S_1(x_\nu - x_\rho) \tag{3.16}$$

3.5.2.1 Implementing FLIP

Focusing more on the implementation aspects of FLIP, Ghourab [2011]'s results proved that FLIP produced much higher resolution compared to pure grid-based simulations. Improvements on performance are also visible in the amount of CPU usage and number of outputs per time step, as well as adaptability of the effects due to the "implicit nature of the particles". Furthermore, Lundberg [2012] states that the modification and overall control of the volume velocity field and particle velocities is an important benefit of using the FLIP approach. Additionally, Calvimontes [2013] concluded that when FLIP was used for his

experiment on wetting a textile the interactions between the fluid body and the solids were satisfying and consistency existed between the solutions of the equations regarding the motion of the grid and the motion of the particles. Salomonsson [2011], proved that when a "look up" is required for neighbouring particles, FLIP can return fast results without going through all the particles since the particles are stored on the grid and all the spatial information are preserved.

Chapter 4

Implementation

The implementation of solver for the presented project was developed in Houdini utilizing the microsolver nodes that are mentioned in the Introduction and allow the developers to setup a solver from zero. Setting up a microsolver network inside DOPs, the simulation will run from left to right and from top to the bottom. According to the nature of the solver and the object attributes, the multiple branches or stream of the network will solve a set of algorithms over a given time.

The reason why Houdini was used to implement the project was mainly because of the ability to dive into the network structure and manipulate it accordingly for our needs. Furthermore, the integrated Vector Expressions (VEX) programming language[SideFX, 2016b], which can be written as a function based program or in network visualisation/graphic representation in VOPs, has been proved to be very powerful due to its highly optimized nature for handling geometry [Garcia, 2016]. It performs likewise to a compiled C++ program, and in some occasions it runs even faster.

Houdini already provides built in solvers that can be used when simulating fire, water, or smoke. The Pyro tool, examined by 1, produces detailed realistic results but on the other hand artists may find it difficult to control entirely the simulation due to lack of comprehension. In some cases, when even higher detail and resolution was in need, the simulation would become extremely heavy requiring large amounts of secondary storage. Finally, the embedded FLIP solver in Houdini is able to simulate hundreds of thousands of particles on the grid that result into visually pleasing results.

4.1 Implementation breakdown

In this section, a detailed explanation of the approach followed to implement the solver and ultimately simulate the fluid motion is presented. A walkthrough the DOPs network and each node connected to it will be explained in order to justify and connect it back to the theoretical and mathematical aspects of the

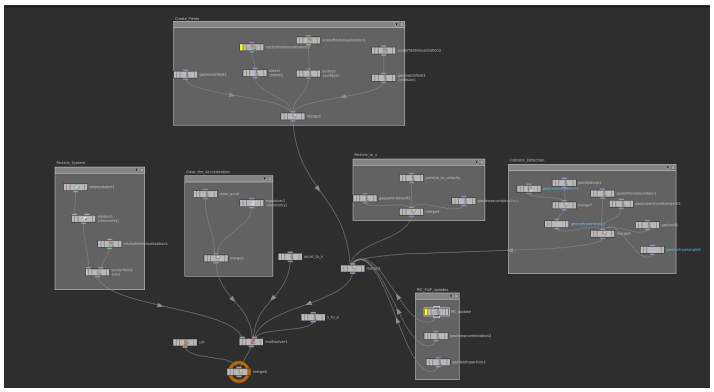

Figure 4.1: The main simulation network inside DOPs

project.

4.1.1 Setting up and initializing the particle system

To start with, the first thing that needs to be done is to setup the particle system that will carry the quantities of the fluid body and will handle advection. The Empty Object DOP node is in charge of this in which we can specify the object that we want to simulate and its state initially in the simulation. An empty container will be created that can have attached to it different types of data [SideFX, 2016a]. It has a unique object name which is guaranteed by the expression obj$OBJ and it has a value of zero when the simulation commences at $ST == 0. To fill this container with the appropriate data, its output will be connected to the input of a SOP Geometry node which is used in order to explicitly create the geometry to be simulated. This node allows the developer to dive inside as if it is a SOP network and model the object or reference the path to a SOP object already created in SOP level without any memory overhead since the object is not copied, only referenced [SideFX, 2016a].

Diving inside the SOP Geometry network, a box (i.e. a square polygon) is created and connected to the Points for Volume SOP. It is important to note that since this implementation is not requiring a continuous stream of particle fluid, the use of a geometry volume is more appropriate rather than flat objects used for emission [Spicer, 2015]. This node is in charge of transforming the geometry into a volume filled with points (i.e. the particles). The final results uses slightly over 160,000 particles since reducing this number or significantly increasing it would result in gaps or noise respectively. It is extremely vital since here is where the configuration of the particles to be generated is explicitly declared. We used the Grid which places the points on the vertices of the 3D grid so that our FLIP algorithm will start taking shape having both particles and a grid. Furthermore, the Point Separation parameter is used to define the smallest distance between the points and the bigger it gets the less particles are

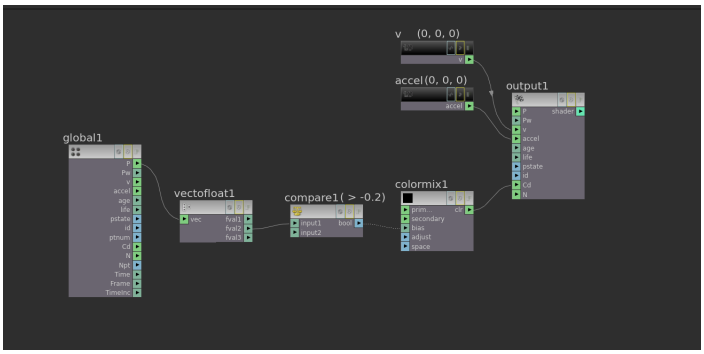

Figure 4.2: VOP network inside the Colour function

generated inside the geometry container. Finally, the particle scale is defined here and added on the geometry as the attribute pscale of type float. The psacle attribute can alternatively be modified by appending a Point SOP in the network and either declaring explicitly the float number or use the expression

$$f@pscale = fit01(rand(@id), 0.1, 1) \tag{4.1}$$

which will randomise the scale.

Inside the SOP Geometry is where the colour discretization will take place. A VEX Builder SOP is appended after the volume is created that will reference the node created in VEX level as a SOP Type. The SOP Type allows the user to build VOP networks that represent the programming language into nodes and manipulate the different attributes of the object, for example the position, velocity or colour and in general any attribute that exists in the geometry structure. In this case, the position is turned from a vector into three float values so we can manipulate them accordingly. The Y value is needed to be compared with a float to determine the position of the particles from the origin. By adjusting this value the difference in the two coloured parts of the volume changes since it influences the bias of a Colour Mix that is appended to it. The changes are eventually connected back to the Cd attribute of the geometry which represents the colour. At this point, the attributes of velocity v and acceleration accel are created both of type Vector. The VOP representation can be transformed easily into piece of code which is represented in the Appendix while Figure presents the graphical VOP network.

The next thing that needs to be done is adding a Vector Field, which according to SideFX [2016a], is "an axis-aligned box divided into individual voxels" that each return a 3D vector that can represent a force direction, in our case the velocity vel. It contains the division size of 0.1, which is approximately the same as the particle division size, and is the value of dynamically resizing the size of the voxels to indicate where the particles are. During the implementation, we have found out that when a bigger number is proposed to the Division Size like

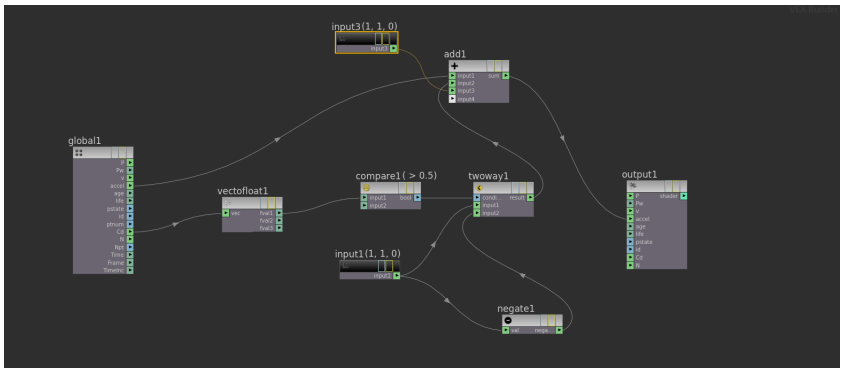

Figure 4.3: The acceleration VOP network

for instance 0.9 instead of 0.1 the fluid moved more uniformly with large numerical dissipation. On the other hand, on extremely low sizes the simulation would crash due to the explosive behaviour. For testing purposes and to determine the behaviour of the field, the microsolver Vector Field Visualization can be used as a second input in order to visualise the field in the simulation.

At this point the particles have the attributes of position, scale, velocity and colour. Since this is the geometry to be processed by the solver and follow all the steps of the fluid equations, it will be fed into the Multiple Solver at the very end of the network. This node is what makes the simulation execute, since it integrates all its attached components in the correct order of each step at the right time and has two inputs: one for the object to be processed and the other for all the rest of the network. The input operators' order of merge is extremely important when connecting the rest of the network to the Multiple Solver, since a wrong order may result to unwanted outcomes or even no change in flow at all.

4.1.2 Clearing and building to acceleration field

The first solver into the second input of the Multiple Solver is in charge of clearing the acceleration at every frame. This can be achieved by using the Gas Linear Combination node which is a microsolver used for combining multiple fields together and it is used in heavy fluid simulations. As we will see at a later point of the breakdown of the implementation this node will also be used to pass the solved velocities from the grid back to the particles. For clearing the acceleration, the Destination field needs to specified as accel and the Geometry field set to Geometry (what we have already set as the geometry to be processed). Since we want for the acceleration to be wiped out for each frame we do not explicitly set any sources for the combination operator and this will result in turning them equal to zero[Lait, 2012].

The acceleration field is built inside a SOP solver which is a combination between the DOP simulation we have built up to this point and SOP nodes

operations to change the state of the object over the given number of frames. Diving in the SOP solver, we create a new VEX Build node similar to the colour one specified earlier. This time the node will create the acceleration field based on colour. Consequently, we extract again the vector into a float and compare the X value of the RBG vector with a float to discretise the colour difference set by the earlier colour VEX node. For each of the two parts of the coloured particles we can establish towards were the particles will move, i.e. their acceleration path. The Two Way Switcher after the comparison serves as the "else" condition. In this way we create two different inputs for each coloured part of the fluid body to manipulate them accordingly. For example, if we wanted the blue particles to move towards the y axis and the white particles towards the x we would explicitly set the acceleration vectors to (0,1,0) and (1,0,0) respectively. All of the above are eventually added back to acceleration of the output and the acceleration force is now created based on colour.

The Gas Linear Combination node for clearing the acceleration and the SOP Solver are combined together with a Merge before been fed to the Multiple Solver. The merge is used to create the relationships between the two nodes and once again the order of merge has impact on the fluid flow.

4.1.3 Adding acceleration to velocity

By using again the Gas Linear Combination node, the acceleration will be passed to the velocities at the third input of the Multiple Solver. With setting the destination field to v we are defining that the field we want to store the result of the operator in is the velocity. The combination operation is defined like the following

$$coef_1 * val_1 OP\, coef_2 * val_2 OP\, coef_3 * val_3 \tag{4.2}$$

and for this node we will use an ADD operator. We set the two sources to v and accel and we multiple the acceleration by the timestep so it will scale dynamically by it.

4.1.4 Old velocity vector field (Beginning of the FLIP implementation)

At this step we start forming the solver in that way so it will meet the FLIP criteria, namely to pass the quantities from the particles to the grid, solve them on the grid and then pass them back to the particles as we have presented in Chapter 3.

We start by using the Gas Resize Field which changes the size of a specified field by adjusting it on the bounding box of the geometry and also can determine the number of voxels by which the bounding box will grow, i.e. resolution (in this case (3,3,3)). The purpose is to track in this way the motion of the fluid "without having to create a field as big as the total possible fluid volume" [SideFX, 2016a].

Hence, we want to resize the previously created velocity vector field vel since this is where the particle velocities, referenced from the Geometry, will be copied into. To do the latter, the Gas Match Field is used to "rebuild fields to match in size and resolution to a reference field"[SideFX, 2016a]. Thus, it will create the vector field oldvel which will match what we previously defined as vel. This is created to justify the step on the FLIP algorithm at which we subtract the old grid velocity from the new and solved one to avoid overwrite and smoothing behaviour which is what the PIC approach produces.

4.1.5 Surface and collision scalar fields

In order to create the "free surface" of the fluid body, namely the boundary condition of the particle system, which will be used on a later point to enforce the incompressibility condition of the Navier-Stokes equations we use again another Gas Match Field. This time it will be a scalar field named surface matched to vel.

Following the free surface field, another scalar field is created to represent the collisions. It is matched on the free surface and will be used to specify the collision field of the simulation when creating solid boundaries later on the network.

4.1.6 Particle To Signed Distance Field (SDF)

Following the creation of the surface scalar field the particles can be converted into a Signed Distance Field using the Gas Particle to SDF node referencing the Geometry. The node uses the pscale of the particles to build the surface field using the Closest Particle Uniform Radius method. The radius value is the same for all the particles and for this method, each voxel is set based on the distance to the closest particle which its radius overlapping the position. It is considered as a fast algorithm for handling spatial lookups.

After the surface is created and initialised, the simulation needs to copy the particle velocities into the velocity vector field vel, based on FLIP, using a Gas Particle to Field node. Hence, we set the destination field to vel, since this is where we want to copy the particle velocities and the particles will be imported from Geometry. As initialized earlier on the simulation, v represents the particle velocities so this is explicitly defined and will be passed to the velocity vector field. Since SDF is used, the extrapolation needs to be carried out at this point in order to check the case where a voxel cell is outside of the particle's radius and so it will use the value of the nearest one. If the extrapolation is not performed at this point, the flow will be significantly more violent. The maximum cell dimension will be multiplied by the maximum extrapolation cell (in our case set to 2). This will be used as a distance cap for "how far away from the particle system that extrapolation should occur" [SideFX, 2016a] and it will fill gaps in the particle system without the need to extrapolate for the entire voxel volume. Increasing the extrapolation would produce cohesive results and

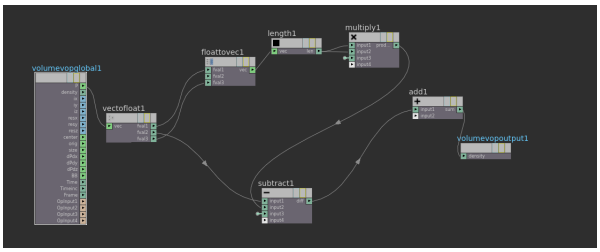

Figure 4.4: The VOP network create the collision field (flat plane)

finer detail [Ghourab, 2011]. Finally, at this step we need copy the vel field to the oldvel field using a Gas Linear Combination.

4.1.7 Boundary conditions and collision handling

Before moving to the last part where the velocities from the grid are updated back to the particles, the collision conditions need to be set. At this branch of the network, the pressure incompressibility condition is also solved. To build the fields that would collide with the fluid, two approaches were adopted. First, by using a Gas Field VOP node and second by using the POP Collision Detect node in combination with a Geometry Wrangle inside of which we can write VEX code.

Starting with the first one, the Gas Field VOP allows us to run VEX on the fields that we have already created and it is preferable when the Gas microsolvers do not satisfy the developers for their purpose. The VEX source is set to "Myself" to refer to the VOP network that is built inside the node. In this case we set a binding between the collision field and the density parameter. In this VOP network (Appendix for the code), the length between the X and the Z value is calculated and squared in order to get a radius on that point. The Y value of the position vector is then subtracted from the radius and the difference will be passed to the density output, and two variables will be accessible from on DOP level; the radius scale and the height. In our case, we zeroed out the radius in order to have a flat plane and the height is set to 2 and it represents where the fluid will hit the plane and change direction accordingly (Figure 4.5)

The second method for collision is used in order to refer to objects from SOP level, and in this project a lighthouse [TF3DM, 2015], a bridge [TF3DM, 2011] and an old farm house [TF3DM, 2014][TF3DM, 2014] were imported to serve as the collision objects. Three nodes are setting and controlling these conditions; the POP Collision Detect node, the Geometry Wrangle node and a Static Object node merge at the end of the simulation with the Multiple Solver. The object, for example the lighthouse, is imported as an .obj into SOP level and transformed into an SDF volume using the IsoOffset surface Node.

The latter, builds an implicit function from the appended to its input geometry and uses the function to create the volume primitive. The Uniform

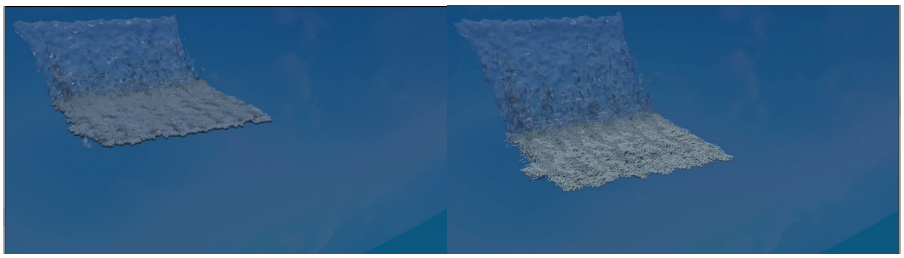

Figure 4.5: Dam break scenario taken from the same perspective but crashing at a different height

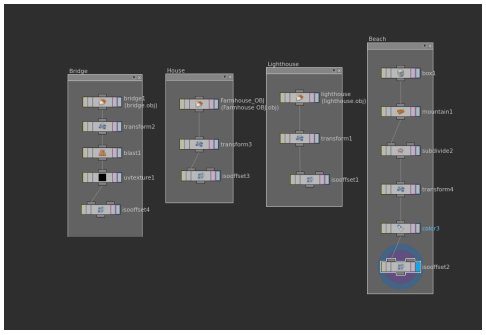

Figure 4.6: The four testing objects all presented in the same geometry

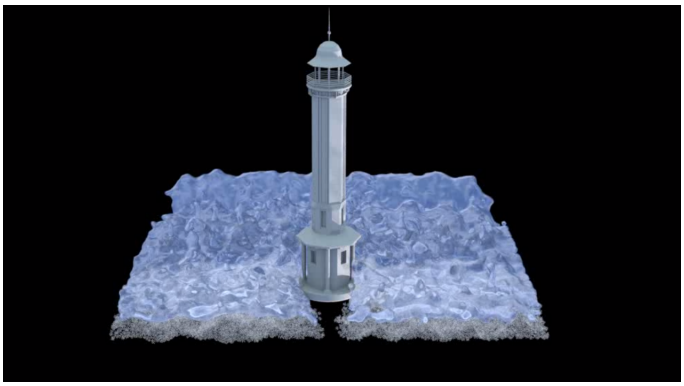

Figure 4.7: The lighthouse test. When the fluid collides with the solid object, the velocities are changed and result to move around it

Sampling is adjusted accordingly to feel all the gaps of the create volume and improve in that way its resolution by ensuring the voxels are cubes. The IsoOffset node will then be referenced inside DOPs on the Static Object sdf on the Proxy Volume field to perform the volume based collision detection. Eventually, the static object will be referenced to the POP collision Detect node which we use for adding to the particles the attribute hitnum, which equals to 1 whenever it hits the static object or 0 when is not. The geometry wrangle then will be utilized to right the following VEX code:

$$if(@hitnum == 1)v@v = -v@v * 0.7; if(v@v.y < 0)v@v.y = -v@v.y; \quad (4.3)$$

This indicates that in the case of the particle hitting the static object, the velocity at that point will be multiplied by -1 to change direction and also multiplied with a float between 0 and 1 known as the "damper". Furthermore, the second condition checks if the Y value of the velocity is negative (due to the upside down camera), and turns it into positive, similar to the absolute value. The addition of the second condition is due to some testing done during the implementation were some particles would exceed the boundary and were passing through the plane. By changing their sign, the collision condition was ensured and the particles behaved like the rest.

The benefit of this setup for collision handling is that an artist would be able to add an object to the SOP level and be readjusting the volume uniform sampling and the damper value and get an equally good result with the lighthouse and bridge that have been tested here.

The reason why the objects were tested was to ensure that the collisions would work in multiple scenarios. The lighthouse(Figure 4.7) is tube shaped base, while the bridge(Figure 4.8 and Figure 4.9) has two columns in the middle that the fluid body should avoid, and the farm house(Figure 4.10.) has the largest width as well as been recorded from a different perspective.. In all

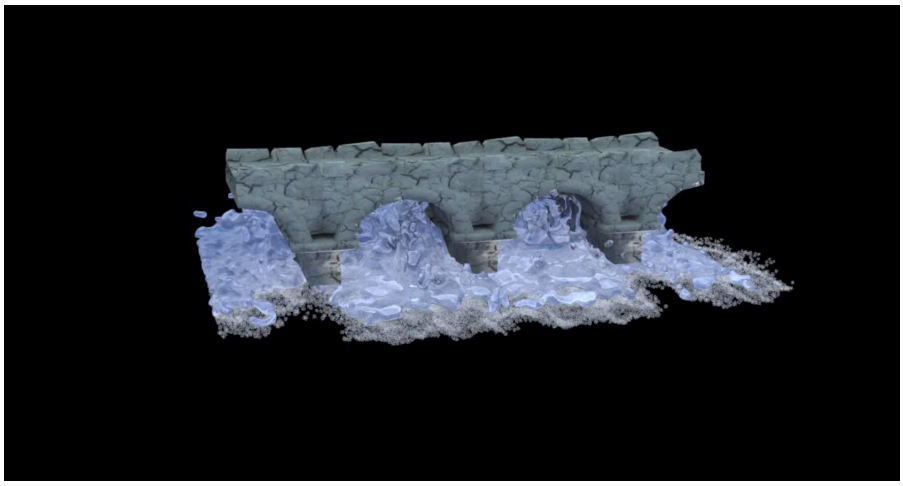

Figure 4.8: The bridhge test. Direction and velocities were distributed between the two arc and the sides and met eventually again

scenarios the fluid successfully went around the object avoiding to flow through it. Additionally, the beach scenario(Figure 4.11) was created to test collision behaviours with a deformed grid that meets the fluid's way when it lands. The results did not unfortunately meet the criteria to be fully successful, but with some refinements would definitely be.

The two approaches for handling the collisions were proved to be equally successful, even though the first one that uses the VOP network had limitations in the shapes that would be created for collision. On the other hand, as already mention the second approach can import in the simulation any kind of object with obvious readjustments. However, another node that was used to ensure the collision handling was done correctly is the Gas Enforcement Boundary, since without it the solver will consider that the velocities inside the object are the fluid's velocities during extrapolation. Hence, this node will enforce the condition by turning into zero the velocities of the collision object. The idea is that the particles within the surface field are already resolved so they can be equal to zero. This is explicitly defined by using the velocity field vel and the collision scalar field collision.

4.1.8 Incompressibility condition

At this subsection, the pressure equation is solved using the Gas Projection Non Divergent node, which removes the divergent components of the velocity field which are causing the expansion or contraction. The way this is handled is by calculating the pressure field that negates compression and apply that field on the simulation at the very moment. Having setup all the required fields earlier on the simulation, they can now be passed on the node to solve the issue.

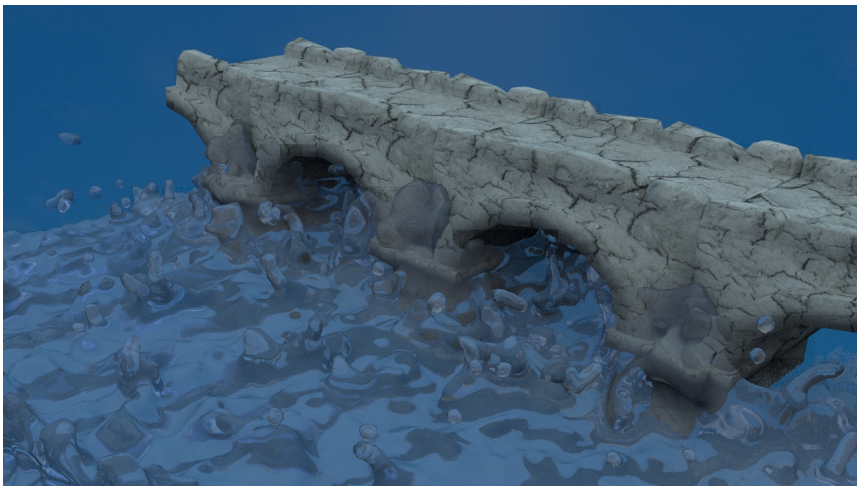

Figure 4.9: The bridge test from the angle behind the flow demonstrating collision detection

Figure 4.10: The farm house is an important test since the new perspective and the increase in width still satisfied the collision detection conditions

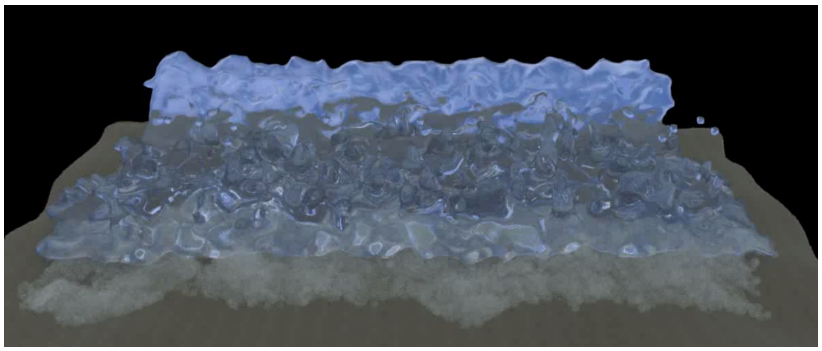

Figure 4.11: The beach test appeared to be breaking before the fluid hit the ground. It was a test to examine how the fluid would continue to move over a deformed plane

IMAGE OF NON DIVERGENT PROPERTIES
According to 15, we can explain the above in simpler words:

- The velocity vector field is the one to become divergence free
- The SDF specifying which voxels are included in the divergence calculations, since incompressibility is only enforced on the interior voxels
- The velocity values on the collision objects are considered fixed, since we zeroed them out
- The scalar field to match The pressure that is required to meet the divergent-free condition
- By preserving the bubbles, the fluid is ensured not have a high freedom factor regarding its fluid preventing numerical dissipation

4.1.9 Particle dismissing

The final part of this branch of the solver is added in order to dismiss particles presenting explosive behaviour and add extra control. This is a known issue when using FLIP and as stated earlier adding partly some PIC aspects to the algorithm may help. Additional to that, the Gas Limit node will secure the velocity vector field with bounds. By setting a maximum value field, the velocities are limited to flow in between specified boundaries. The node is not extremely valuable to the solver, but during tests it proved to be relatively helpful for controlling the overall body of the fluid.

What proved to be more important was a Geometry Wrangle that uses the removepoint function from VEX (Appendix). Going through some tests, and observing the behaviour of some particles, we managed to remove them by explicitly checking their position values on X, Y, and Z. Even though a trivial

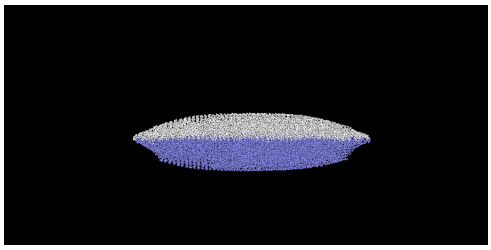

Figure 4.12: The first results from the testing of the PIC update

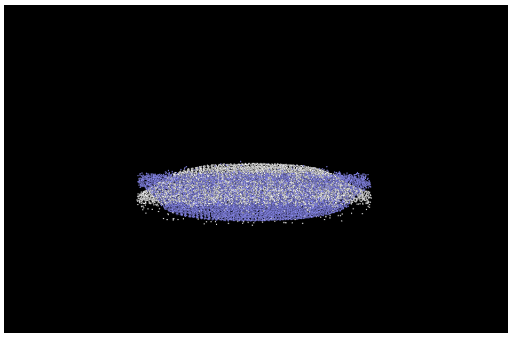

Figure 4.13: The first results from the testing of the FLIP update

way to enforce particle remove, it helped the solver in a positive way giving smoother and more realistic results and minimized the noisy behaviour FLIP is known to be producing.

4.1.10 PIC update, subtracting velocities and passing velocities back to particles

The final step requires the transfer of the velocities back to the particles. The PIC update approach is achieved through a Gas Field to Particle which is in charge of copying the field values into a point attribute of the geometry. The resulting values will overwrite the particle attributes as described earlier (vel copy to v).

The FLIP update is handled by two nodes. First the Gas Linear Combination will subtract the old velocity field from the velocity field. Then, the difference will need to be transferred back to the particles by using the Gas Field to Particle. In this case the calculation will be Add instead of Copy which we used on the PIC update and this satisfies the FLIP algorithm. Figure 3.1 is justified and implemented correctly and is fully functional since the difference in the particles' behaviour is visible.

Finally, a Gas Linear Combination will pass the velocities from the grid back to the particles positions, hence the operation will be an Add between P and v

and the velocity will be multiplied by the timestep.

4.2 Refinement in SOP Level for visualisation

In this section some final refinements that have been made are explained in order to improve the overall result of the solver. These refinements were appended to the simulation node in SOP level.

An Attribute Wrangle is used to set the colour of the particle system to blue. Since the particles are already divided in two colours, light blue and white, to control the acceleration and visualise the foam, this node takes the Y value of the position and adds an offset float value. If the Y value of the position is less than the sum of the Y value and the offset, the colour attribute Cd is set to blue. This will result in the simulation to turn the whole body into a uniform colour and when reach a specific point return to the original ones specified inside the simulation. This approach was taken so that the foam would be explicitly separated from the main body of the fluid and have more realistic visual results rather than starting the simulation with the foam on top already. This node serves as the criteria that need to be met in order for the fluid to create foam. In more complicated simulations, this would be handled inside the simulation during the interpolation step or when custom conditions set, but due to time limitations and more focus on the colour function this approach was not followed.

The following two nodes, an Attribute VOP and a Python node, are extracted from the simulation assignment that was completed last May **[Agrotis, 2016]**. Since they are not a priority for the correct functionality of the simulation (i.e. they can be bypassed without influencing the simulation) their description here is not extended. The only reason they were added was to add some additional liveliness or turbulence (i.e. similar to a wind external force on the surface) in the overall fluid motion for visualisation purposes but not influence in a big way the velocity advection. The assignment focused around the area of ocean surface simulation and followed a simple approach of combining Perlin noise with the sum of sine waves, an idea introduced by Max [1981]. The Attribute VOP simply adds noise on the Y value of the position vector P and promotes the parameters of frequency, offset, amplitude and roughness.

- Frequency controls the sparsity of the noise
- Offset controls the direction
- Amplitudes represents the height of the noise
- Roughness control the smoothness

The Python SOP serves for introducing the sine formula to the overall movement by transforming the overall geometry shape with the Perlin noise on top of it. Additional to frequency and amplitude that have the same functionality as

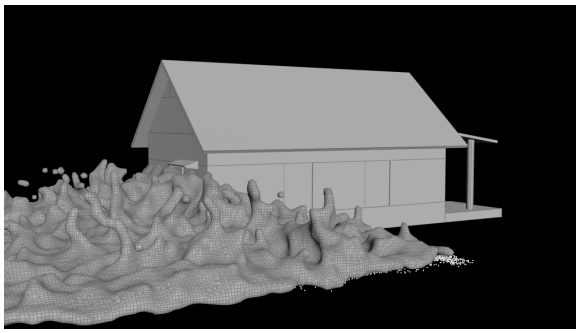

Figure 4.14: Particles transformed into surface

previously described, the Python SOP adds the phase parameter which controls the number of curves the sin wave has. (Appendix)

The following Delete node acts as the discretization factor between the initially set white (for foam) and light blue colours of the particles. By using the "Delete by Expression" filtering, we used

$$if(CR >= 0.35, +1, 0) \tag{4.4}$$

in order to get the light blue values. This is the breaking point where the white values would be imported from the simulation to a new Geometry node in object level and use the same Delete node but reversing the selection condition (either by stating "Delete Non-Selected on the Operation menu, or by transforming the greater or equal than operator to less or equal than).

For further visual refinements, further Delete nodes were used for testing the simulation and the overall behaviour. An Attribute Create node was used to sum the absolute values of the velocity, and following down the network delete points that would not meet specified conditions. The same fashion was used on the Foam geometry.

4.2.1 Particle Fluid Surface, Shading and Lighting

Even though the main focus of the project was to simulate the fluid following the correct mathematical approaches, an amount of time was also dedicated towards shading the water and creating more visually pleasing results. Rendering liquid effects requires a lot of skill due to its reflective and refractive nature (Belyaev SIM). In a FLIP simulation, the resolution is defined by the size of the grid, the number of particles, the scale of these particles and the extrapolation distance [Ghourab, 2011]. The Particle Fluid Surface node creates from a given particle system a surface (image comparison). It is an essential node that needs to be used when we are in need for shading the water surface.

The Basic Liquid shader from the default Houdini Shop was used for shading the main body of the fluid. Additionally, the Volume Cloud shader combined

Figure 4.15: Compared to Figure 4.5, in this case the foam is shaded

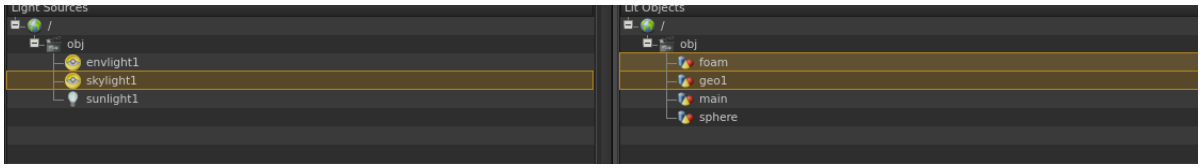

Figure 4.16: The Light Linker environment in Houdini. In the case of the highlighted objects the light reflects while the rest are not lit

with Sprays was used for the foam. In the videos submitted, one can also see the case where the foam is not shaded but the particles appear with smaller pscale to create the sense of foam.

Speaking also about some differences in the videos, some of them have used a sphere to surround them that reflects the environment map for further refinement of the liquid shader. It was shaded by using a simple mantra surface of a constant colour. Even though it gave more realistic results and transparency on the water, it was not well received by the foam and that's why the main result tests are considered as the ones with the black background.

Finally, the lighting was achieved by an Environment light using a sky HDR Map [Bzfusion, 2005], as well as sunlight and a skylight. The last two lights are not reflecting on the main body of the fluid since they would create large amounts of reflection or refraction, hence the environment light combined with the liquid shader are in charge of the result regarding the fluid. Light reflection exclusion can be manipulated through the Light Linker Pane.

Chapter 5

Conclusion

This thesis presented the idea of implementing a FLIP fluid solver in the Dynamics network in Houdini. The objectives of this project that were achieved are

- Realistic enough visual results been driven by a well defined simulation network.
- Checked the simulation with multiple solid objects to enforce boundary conditions
- Create the foam with an easy approach of the colour function and split the main body from the foam body without recomputing velocities or introducing a second particle system
- Follow the FLIP method correctly

The results are pleasing and quite accurate as it can be seen from Chapter 4. The approach of setting up the FLIP solver from zero, using the microsolvers and VEX functions was really important since the author focused on both technicality and mathematical complexity. The parameters of position,velocity and colour were successfully simulated as have been proven by the early tests on Figures 4.12 and 4.13 with the PIC update differing from the FLIP update in the terms of visualisation.

The collision detection testing has also proven that the simple collision detection algorithm is actually working on various types of shapes and redistribution of the fluid has successfully be done. Furthermore, the collision simulation can handle larger amounts of particles with the main tests having a particle system consisted of approximately 160,000 particles.

5.1 Future Work and Improvements

Definitely, the implementation is not perfect. A problem that was apparent from the first moment was the violent behaviour of particles in the FLIP solver. Due

to the fact that the initial idea was to create a tool for simulating an overflowing dam, it took quite some time for this idea to be dropped so the the explosive behaviour was for desirable in the beginning of the implementation process. This might have been tamed with the introduction of the particle dismissing approach, however is not necessarily correct. With more time under our belts, a better handling of this particles could be done for example by involving their age and whether they are dead or alive on the system.

Another improvement would be to test the simulation with more bodies of fluid. Even though it works correctly with solid objects and boundary enforcements, the simulation is not tested at all with other bodies of fluid or even animated objects. Furthermore, additional forces other than acceleration of gravity could be introduce, like for example buoyancy. By examining the buoyancy, tests with floating objects would take place to justify like for example a ship or a plastic duck.

Moreover, even though the setup of a tool is not that necessary for this project since it was mainly for researching purposes, a Digital Asset could be trivially created allow more control over the network of the DOP.

Finally, the last thing that could change is the way of addressing the issue of the foam. The colour function might be convenient and computationally light, however a different interpolation of particles could be followed.

At this point, where the simulation is currently standing, a broad understanding of fluid simulation was acquired and additionally of dynamic operators in Houdini. It was an overall challenging and intellectually stimulating experience, since no previous knowledge of the used node set was present.

Bibliography

Alexis Agrotis. *Simulation techniques project: Ocean simulation.* PhD thesis, Bournemouth University, 2016.

Jeremiah U. Brackbill, Douglas B. Kothe, and Hans M. Ruppel. FLIP: A Low-Dissipation, Particle-In-Cell method for Fluid Flow. *Computer Physics Communications*, 48(1):25–38, 1988.

Robert Bridson. *Fluid Simulation for Computer Graphics.* CRC Press, 2008.

Bzfusion. HDR Enviroment Map, 2005. URL `http://www.bzfusion.net/skymaps/sky_lightblue.jpg`.

Alfredo Calvimontes. Simulation of Textile Wetting Using Fluid Implicit-Particles (FLIP). In *46th International Detergency Conference, At Düsseldorf*, pages 1–11, 2013.

Peter Claes. *Controlling Fluid Simulations with Custom Fields in Houdini.* PhD thesis, Bournemouth University, 2009.

Jens Cornelis, Markus Ihmsen, Andreas Peer, and Matthias Teschner. IISPH-FLIP for Incompressible Fluids. *Computer Graphics Forum*, 33(2):255–262, 2014.

Fernando de Goes, Corentin Wallez, Jim Huang, Dmitry Pavlov, and Mathieu Desbrun. Power Particles: An Incompressible Fluid Solver Based on Power Diagrams. *ACM Transactions on Graphics (TOG)*, 34(4):50–61, 2015.

Deborah R. Fowler. Fluids, 2013. URL `http://deborahrfowler.com/HoudiniResources/WriteUps/Fluids/FluidsIntro.pdf`.

Beau Garcia. Creating Custom Houdini Solvers With VEX Wrangles, 2016. URL `http://www.digitaltutors.com/tutorial/2433-Creating-Custom-Houdini-Solvers-With-VEX-Wrangl`

Ahmad Ghourab. *A Fluid Implicit Particle Approach to a Pyro Solver in Houdini.* PhD thesis, Bournemouth University, 2011.

Francis Harlow. The Particle-In-Cell (PIC) Method For Nuerical Solutions of Problems of Fluid Mechanics. In *In Processings Symposium in Applied Mathematics*, volume 15, page 269, 1963.

Ioannis Ioannidis. *3D Particle In Cell/Fluid Particle Fluid Solver using OpenMP directives*. PhD thesis, Bournemouth University, 2012.

John Johansson. *Efficient Implementation of the Particle Level Set Method*. PhD thesis, Linkoping University, 2010.

Jeff Lait. Building Fluid Solvers From Scratch, 2012. URL `http://archive.sidefx.com/index.php?option=com_content&task=view&id=2200&Itemid=344`.

Michael Lentine, Matthew Cong, Saket Patkar, and Ronald Fedkiw. Simulating Free Surface Flow with Very Large Time Steps. In *Proceedings of the 11th ACM SIGGRAPH/Eurographics Conference on Computer Animation*, pages 107–116. Eurographics Association, 2012.

Lukas Lundberg. *Art Directed Fluid Flow With Secondary Water Effects*. PhD thesis, Linkoping University, 2012.

Nelson L. Max. Vectorised Procedural Models for Natural Terrain: Waves and Islands in the Sunset. *ACM SIGGRAPH Computer Graphics*, 15(3):317–324, 1981.

Keth Museth. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Transactions on Graphics (TOG)*, 32(3):27, 2013.

Fredrik Salomonsson. *PIC/FLIP Fluid Simulation Using Block-Optimized Grid Data Structure*. PhD thesis, Linkoping University, 2011.

SideFX. Houdini Help Cards, 2016a. URL `http://www.sidefx.com/`.

SideFX. VEX, 2016b. URL `http://archive.sidefx.com/docs/houdini15.0/vex/_index`.

Phil Spicer. H14_Dynamics_MSc. Technical report, Bournemouth University, 2015.

Jos Stam. Stable Fluids. In *Proceedings of the 26th Annual-Conference on Computer Graphics and Interactive Techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

TF3DM. Stone Bridge 3D model, 2011. URL `tf3dm.com/3d-model/stone-bridge-37857.html`.

TF3DM. Old Farm House 3D Model, 2014. URL `http://tf3dm.com/3d-model/old-farm-house-91130.html`.

TF3DM. Lighthouse 3D model, 2015. URL `tf3dm.com/3d-model/lighthouse-966.html`.

Nils Thürey, Matthias Müller-Fischer, Simon Schirm, and Markus Gross. Real-time Breaking Waves for Shallow Water Simulations. In *Computer Graphics and Applications*, pages 39–46. IEEE, 2007.

Magnus Wrenninge. Fluid Simulation In a Visual Effects Context, 2011. URL `http://webstaff.itn.liu.se/ jonun/web/teaching/2011-TNCG13/Lectures/Lecture03-MW/Wrenninge`

Yongning Zhu and Robert Bridson. Animating Sand as a Fluid. *ACM Transactions on Graphics (TOG)*, 24(3):965–972, 2005.

Appendix

The VEX Code from the multiple VOP networks is presented here:

```
sop
colour()
{
    vector      v1;
    vector      accel1;
    float       fval1;
    float       fval2;
    float       fval3;
    int bool1;
    float       output1;
    vector      clr;

    // Code produced by: v
    v1 = { 0, 0, 0 };

    // Code produced by: accel
    accel1 = { 0, 0, 0 };

    // Code produced by: vectofloat1
    vop_vectofloat(P, fval1, fval2, fval3);

    // Code produced by: compare1
    bool1 = (fval2 > -0.20000000000000001);

    // Code produced by: autoconvert
    output1 = bool1;

    // Code produced by: colormix1
    if (0)
        clr = hsvtorgb(vop_colormix(rgbtohsv({ 1, 1, 1 }),
                                    rgbtohsv({ 0.29999999999999999, 0.29999999999999999, 1 }),
                                    output1, 1));
    else
        clr = vop_colormix({ 1, 1, 1 }, { 0.29999999999999999, 0.29999999999999999, 1 }, output1, 1);

    // Code produced by: output1
    vector tempv = v1;
    vector tempaccel = accel1;
    vector tempCd = clr;
    v = tempv;
    accel = tempaccel;
    Cd = tempCd;
}
```

Figure 5.1: VOP Color

```
sop
accel2()
{
    float       fval1;
    float       fval2;
    float       fval3;
    int bool1;
    vector      input1;
    vector      negated;
    vector      result;
    vector      input3;
    vector      sum;

    // Code produced by: vectofloat1
    vop_vectofloat(cd, fval1, fval2, fval3);

    // Code produced by: compare1
    bool1 = (fval1 > 0.5);

    // Code produced by: input1
    input1 = { 1, 1, 0 };

    // Code produced by: negate1
    negated = -input1;

    // Code produced by: twoway1
    #ifndef _vex
      result = (0 != 0 ? (bool1 != 0 ? negated : input1)
                       : (bool1 != 0 ? input1 : negated));

    #else
      #if !strcmp("v", "b")
        float scale = (0 != 0 ? (bool1 != 0 ? 1 : 0)
                              : (bool1 != 0 ? 0 : 1));
        result = (1-scale)*input1 + scale*negated;
      #else
        result = (0 != 0 ? (bool1 != 0 ? negated : input1)
                         : (bool1 != 0 ? input1 : negated));
      #endifsop
    #endif

    // Code produced by: input3
    input3 = { 1, 1, 0 };

    // Code produced by: add1
    sum = accel + result + input3;

    // Code produced by: output1
    vector tempaccel = sum;
    accel = tempaccel;
}
```

Figure 5.2: VOP Accel

```
import math

node = hou.pwd()
geo = node.geometry()

# Add code to modify contents of geo.
# Use drop down menu to select examples.

freq = node.evalParm("frequency")
phase = node.evalParm("phase")
amp = node.evalPara("amplitude")

#access the parameter of ramp color
ramp = node.parm("color").evalAsRamp() #return parameter as object

#next we store all the points
points = geo.points()
cd = geo.addAttrib(hou.attribType.Point, "Cd2", (1.0,1.0,1.0)) #add the attribute of color, 1 1 1 default value of vector

#first create a list with only the white values
ypos = [p.position()[1] for p in points] #a way to create a loop and I am getting the value I want to store = the y position
miny = min(ypos) #set the lowest point on y before deformation
maxy = max(ypos)

#height variable
height = maxy - miny
v = 1.0/height #store the value to multiply the position wave.

#set the sin wave
for p in points: #p is a single point
    pos = p.position() #getting the position of that point
    x = pos[0] #0 will give us x. pos is a list with three values x,y and z
    z = pos[2] #our wave will be moving along z
    y = pos[1] + math.sin(z*phase + (hou.frame()*freq))*amp #grab the curve position that I already have and add on top of that the sin wave
    #if we delete pos[1] we lose the original deformation. so we keep it and add the sin function on top of it
    p.setPosition((x,y,z))

    #get the look of the value
    lv = (pos[1] - miny)*v #retrn a value between zero and one
    color = ramp.lookup(lv) #get color based on that value
    p.setAttribValue(cd, color)    #set the color attribute
```

Figure 5.3: Python Code from Simulation assignment