Fluid Solver built in Houdini

Sydney Dimitra Kyrtsia

(i7658072)

MSc Computer Animation and Visual Effects

Bournemouth University

NCCA

August 2017

Table of contents

Abstract

Fluid simulation is one of the most researched subjects in Computer Graphics. This paper describes the theory and background research of fluids simulations related to this project. A basic fluid solver is implemented in SideFX's Houdini from the ground up, using its microsolvers and dynamic network structure. Different tests were made using both the PIC and FLIP methods as well as simulations with different collision objects.

Keywords: Fluid, FLIP Fluid Solver, Houdini

Introduction

Modeling and animating natural phenomena is an area of big interest for the Visual Effects community since they cover a wide variety of forms and effects. Phenomena containing fluids have always been a challenge since they move and interact in wide range of ways making it difficult to control and art direct. Usually the fluids are controlled by certain rules and not directly from the artists, which can be quite challenging to produce a certain result especially in scenes where the fluid is exhibiting physically impossible behavior.

Natural looking fluid simulations are highly desirable but difficult to produce because they exhibit various behaviors. This popularity has attracted a vast amount of research over the years making physical simulations of fluids for photorealistic scenes, animation and games, one of the most researched topics in computer graphics.

There is a high demand of control and realism in 3D shots which is challenging, especially by artists that not only need to control the movement but also the overall look of the fluid and its behavior through different parameters.

The complexity of dynamic flows is examined by the field of Computational Fluid Dynamics (CFD). It studies a wide variety of fluids and effects like fire, smoke, gas and water as well as high viscosity fluids and melting. Fluid simulations tools are widely used today in films, games and the entertainment industry in general, which increases the demand of realism, control and speed. This increasingly high demand pushes for more research and continues progress in an already widely covered area.

Although, there is a lot of CFD research available, the solutions and tools that physics and engineering have developed are focused on physically accurate results. In the visual effects industry, the goal is to create a specific look, managing a balance between time and quality.

The aim of this project is to build a Fluid Solver using the Houdini software package implementing a selection of techniques and methods to create realistic liquid simulations.

Houdini Software

SideFX's Houdini Software was chosen for this project due to its procedural tools and because it's widely used in the visual effects film industry. It offers a node based dynamics system which has an open architecture.

It includes fluid solvers and fluid simulation shelf tools. The latest 16th version of Houdini, offers a big variety of pre-built in tools to simulate fire, water and smoke. Artists can use these tools straight from the shelf, control the result by adjusting the parameters provided from the tool's user interface, and all that in a very fast and efficient way.

For cases that the tools do not offer the desired speed, resolution or control, they can be easily improved. Every pre-existing digital asset can be unlocked, studied and modified to support extra functionality in accordance to the needs of the current simulation offering a wide range of control and flexibility over the production pipeline.

Houdini also allows using scripting to implement nodes. The integrated Vector Expressions (VEX) programming language has been proven a very powerful and useful tool for handling particles. It performs like a complied C++ program and it can run even faster. [Agrotis, 2016]

Thesis organization

This Master thesis is organized as follows:

The first chapter begins discussing previous research related to the project. It also provides the reader with the scientific and technical background of fluid simulations and introduces the mathematical equations that govern the behavior of liquids required to understand and implement a basic fluid solver. That is followed by an analysis and comparison of the most common used methods for fluid simulation

The second chapter contains the Design part of the project where the requirements and goals are set. Also, there is a short description of the main Houdini tools

That is followed by the analysis of the solver's network and how it is implemented.

The forth chapter, includes the Simulation examples that use the solver to showcase how it works.

In the last one, an analysis of the results is included with a review on the known issues of the solver. Finally, ideas for further study and improvement are presented.

1. Previous Research

Related Work

Fluids have been the subject of many research papers, publications and lectures both from the scientific and visual effects community. This chapter includes a summary of the previous work that studied for the purposes of this project.

There is a range of methods and approaches that were developed from the early stages of simulating fluids: from the first fluid simulation methods described by Harlow [1962] where they used the Particle-In-Cell (PIC) method, to Gingold and Monaghan, and Lucy [1977] inventing the smoothed particle hydrodynamics (SPH) in order to simulate astrophysical phenomena.

After that computer graphics started using particle systems to simulate physical based elements, with Reeves, [1983] introducing a particle system as "a technique for modeling a class of fussy objects". According to Reeves, "particle systems model an object as a cloud of primitive particles that define its volume. Over a period of time, particles are generated into the system; move and change form within the system, and die from the system."

Foster and Metaxas, in 1997 in their paper 'Controlling fluid animation, solved the 3D Navier-Stokes equations to simulate liquids on a regular grid. The Navier-Stokes equations were also used by Mullet et al. [2003] to created water simulations with free surfaces calculating the interaction forces between the particles from SPH.

Zhu and Bridson [2005], in their Siggraph Paper "Animating Sand as Fluid", they describe a method of modifying the fluid solver to simulate granular materials like sand. They developed an algorithm that uses the advantages of both the grid and particle method getting more flexibility and efficiency. (Image 1.1)

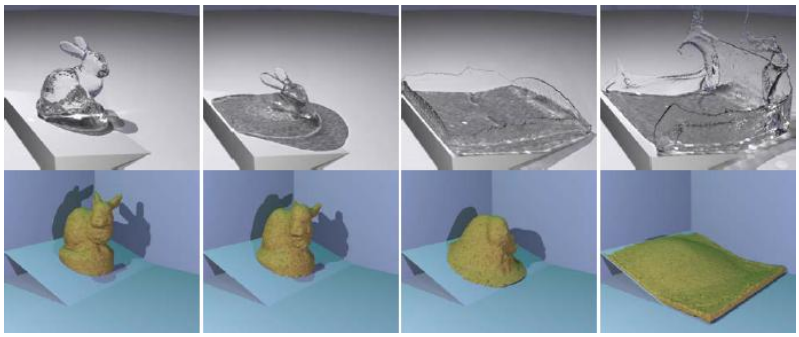

Image 1.1: The Stanford bunny as water and as sand (Zhu and Bridson, 2005)

Horvath and Illes [2007] used the particle-based approach with the aim of creating a physically correct fluid simulation system. In order to render it, they created a Houdini plug in to integrate their simulation engine with the Houdini software.

One of the most important sources of this paper was Bridson's book "Fluid Simulation for Computer Graphics", 2008, because it offers an extensive analysis and comparison of the basic methods used in fluid simulations, which are discussed in a later chapter.

Zhang et al [2015] developed a method for fluid simulations where the fluid is driven to follow or match a moving or deforming target. An example, where the user specifies a path and the fluid flows through it, can be seen in Image 1.2 below.

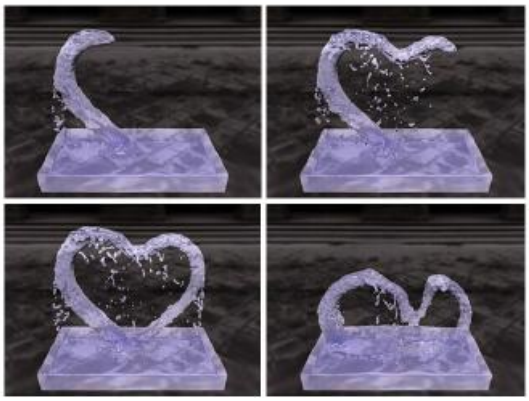

Image 1.2: Path control example by Zhang et al. [2015]

Regarding the Houdini tools and solvers, several studies have been completed with focus on creating new solvers or improving the existing ones aiming for higher efficiency, speed or artistic control. In a previous Master's project, Ghourab [2011] implemented an alternative solver for the Houdini Pyro tool aiming for better performance and speed. Furthermore, Claes [2009] worked on existing Houdini solvers and modified aiming for more artistic control over fluid simulations.

Finally, Agrotis [2016] implemented a FLIP solver in Houdini using the basic nodes to solve the Navier-Stokes equation and produce realistic looking water waves that collide with different objects or fall from a certain height. The solver described in this paper follows a similar "minimalistic" approach of a FLIP fluid solver that Agrotis [2016] has implemented, and expands it trying to address and solve some of the problems occurring, such as control of the viscosity and incompressibility of the fluid.

Priscott [2010], Perseedos [2011] and Moorhead [2016] (Image 1.3), used the SPH method to implement a fluid system using C++ and OpenGL. Their simulations were then exported from the C++ and OpenGL application to Houdini for more realistic visualization and rendering.

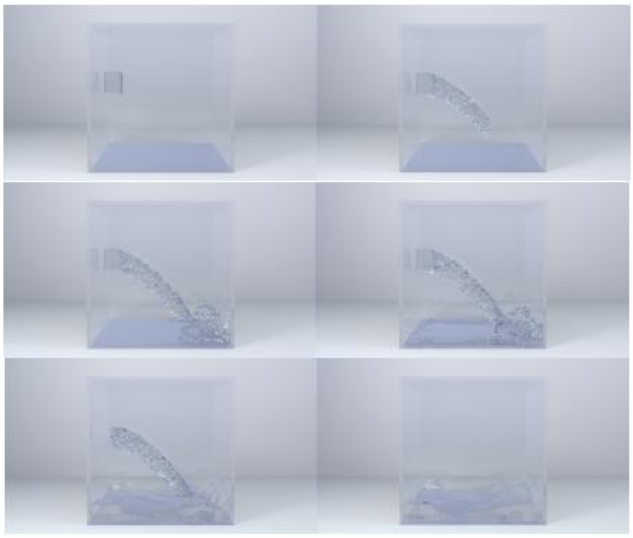

Image 1.3: Water in a Glass Tank. (Moorhead, 2009)

Scientific Framework

This section provides the Computational Fluid Dynamics (CFD) background which includes the Navier-Stokes equations. They describe the motion of liquids, and the physical meaning of their different terms. Fluids include two types of materials which are liquids and gases: like water, smoke and fire.

Fluid Dynamics and specifically its sub-discipline the Hydrodynamics is a complex and wide researched area in Fluid Mechanics that study the movement of fluids and covers many phenomena, parameters and mathematical models. Hydrodynamics deals with conservation laws in fluids, Newtonian and non-Newtonian fluids, compressible and incompressible flow, viscous and inviscid fluids, just to name a few.

This paper focuses on implementing a fluid solver in Houdini in a specific time framework. That required an extensive study of the scientific aspect of fluids. Below there is a presentation of the most important and relevant to the project.

Navier-Stokes equations

The fundamental theory of the fluid motion is based on the Navier-Stokes equations. Navier-Stokes equations are the physical equations that describe the motion of fluids. That is why they tend to be used as a foundation for a lot of solvers in the existing software packages, like Houdini

The Incompressible Navier-Stokes equations are a group of partial differential equations that are being used as the primary model to simulate fluids (liquids and gases) serving as the base of many of the existing solvers.

There are several different methods that solve these equations to create fluid simulations. Some of them are: Eulerian or Lagrangian methods, the Smooth Particle

Hydrodynamics (SPH), Lattice Boltzmann methods and more. The different methods are selected depending on the unique needs.

Mathematically, the state of a liquid at a given instant of time is modeled by means of velocity vector field and a pressure field. The Navier-Stokes equations describe how these two fields are coupled and how they change over time. The equations provide a mathematical model that describes the complex mathematics of fluid motion following the laws of physics. By solving them we get values of the unknown of the fluid, specifically the new acceleration which allows for its new position to be calculated.

The Incompressible Navier-Stokes equations are:

The *momentum equation* describes how the fluid accelerated due to the forces acting on it:

$$\frac{\partial \mathbf{u}}{\partial \mathrm{t}} = -\mathbf{u} \cdot \nabla \mathbf{u} - \frac{1}{\rho} \nabla \mathrm{p} + \mathrm{v} \nabla \cdot \nabla \mathbf{u} + \mathbf{F} \qquad (1.1)$$

And the *incompressibility condition* that ensures that the velocity field has zero divergence, therefore conserving mass:

$$\nabla \cdot \mathbf{u} = 0 \qquad (1.2)$$

Where:

u - is a vector that presents the velocity of the fluid**.**
ρ - stands for density of the fluid.
p - stands for pressure, which is the force per unit area that the fluid exerts on anything.
v - is the kinematic viscosity (measures how viscous the fluid is) (μ/ρ).
F - any external forces that are applied to the fluid.
(*The **vector** fields are presented in bold letters.)

The *momentum equation*

Both equations are mathematical statements of basic conservations laws of physics: the first equation (1.1) derives from Newton's second law and the second (1.2) is an expression of the conservation of mass.

According to Newton's second law:

$$\boldsymbol{F} = m\boldsymbol{a} \qquad (1.3)$$

where the acceleration of each particle is given by:

$$\boldsymbol{a} = \frac{D\boldsymbol{u}}{Dt} \qquad (1.4)$$

(*Vectors are represented in bold letters)
F - represents the external forces.
m - the mass of the object
α - is the acceleration
u - the speed and
t - is the time

The first equation (1.1) states that every change in the velocity of the liquid at a certain position is a result of the four processes described by the four terms on the left side of the equation:

- $(\mathbf{u}\cdot\nabla)\mathbf{u}$: is the (self-)**advection term**, which represents the transport of matter by the fluid's flow, showing how the divergence affects the velocity. Physically it represents the phenomena where the fluid motion causes motion to the entities it includes.
- $(1/\rho)\nabla p$: the **pressure term** (pressure is defined as the force per unit area), which models how the particles move away from high pressure areas. The gradient of a scalar field equals to a vector field pointing in the direction of the greatest 'ascent' of the scalar field. The minus sign in front of the gradient of pressure, points away from the high pressure and towards the low pressure. If the pressure is equal in every direction there is going to be zero acceleration due to pressure. The force due to pressure derives from the imbalance between high pressure areas and low pressure areas (Bridson, 2008).

$v\nabla\cdot\nabla\mathbf{u}$: the **diffusion term**, which captures the relationship between the velocity and viscosity variable v. In other words, the diffusion allows the velocity to propagate outwards, and the viscosity controls the speed of the propagation (Claes 2009).

F : the sum of all the **external forces** that are applied to the fluid (like wind). In the simplified version, where gravity is the only external force **F** can be represented by g. **(g** - is the acceleration due to gravity)

The *incompressibility condition*

The incompressibility condition (1.2) preserves the mass of the fluid throughout the simulation as the divergence of the velocity field equals to zero. The conservation of mass in other words says that the amount of fluid flowing into the predefined volume must be equal to the amount that's flowing out.
The incompressible case is simpler than the compressible one, as the fluid is assumed to be isotropic. For the special case of an incompressible flow, the pressure constrains the flow so that the volume of the fluid elements is constant. Isochoric flow resulting in a solenoid velocity field with $\nabla \cdot \mathbf{u} = 0$

Within Houdini, as it described in the Implementation chapter, the divergent is removed from the velocity field using the Gas Project Non-Divergent microsolver. That is the part of the velocity field that represents expansion or contraction. The "removal" is achieved by computing a pressure field that counteracts any compression and applying that pressure field instantaneously keeping the density constant (SideFX, Help Card).

The Navier-Stokes equations are very hard to solve due to non-linearity the (self-) advection term presents. In fact, an analytical solution which provides the velocity and pressure of the fluid at all points in space and time has not been found yet. (Ausseloos, 2006)

There are two main approaches that provide solutions of the Navier-Stokes equations: the Eulerian and the Lagrangian method. Their main difference lies with the way they store and calculate the attributes. They are both described below.

Eulerian and Lagrangian methods

The Eulerian and Laplacian methods, both solve the Navier-Stokes equations to obtain the velocity of the liquid at different points in space. The different methods are chosen and used depending on the characteristics and requirements of the simulation.

The grid-based method:

In the Eulerian viewpoint, the simulation takes place in a specific finite volume of space, which is represented by a 3D or 2D grid or field. The grid space is divided into a number of rectangular cells and the fluid flows through it. The quantities of the fluid are defined at every point of the grid. Instead of keeping record of every single particle and its attributes, we keep record of the cells of a grid where we can measure how their values change over time. (Bridson, 2008)

The particle based method:

According to the Lagrangian viewpoint, we define the fluid flow with particles where each particle carries its own properties, like mass, velocity, density etc. Conservation of mass and Newton's laws apply directly to each fluid particle. The Lagrangian method is easy to implement as all the calculations are performed on the particles, but that makes them to rely heavily on the density which means in areas with low density, they could develop high percentage of inaccuracy (Strantzi, 2016).

Comparison

Both methods provide the fluid quantities and their change over time, forming a graph of the fluid quantities evolution but their graphs would look different due to the difference in the way they measure the rate of change. Both methods are used for simulations taking advantage of their strong points and handling their limitations, according to requirements of the desired results.

The Eulerian method's biggest weakness is that with the currently available computer processing power, it is limited in size and resolution to the grid's size and resolution respectively. This means that in large simulations that require high resolution, they suffer from "numerical dissipation" which results in mass loss.

In comparison, the Lagrangian viewpoint doesn't have these grid related limitations but its main disadvantage appears to be in the difficulty to form a smooth liquid surface.

Hybrid Viewpoint

In order to overcome the disadvantages, hybrid methods have been developed that use components of both the Eulerian and Lagrangian methods. Usually, fluid quantities are carried by the particles that flow through an Eulerian grid. The advection term is calculated on the particles, and the other quantities are integrated on the grid (Ghourab, 2011).

	Lagrangian	Eulerian
Advantages	• Unlimited simulation size • Can be used for different types of fluid	• Independent performance from the number of particles • Fast • Easier to mathematical analytical work with spatial derivatives (like the pressure derivative)
Disadvantages	• Large number of particles usually required for realistic simulations causing memory problems • Errors due to inaccuracy in areas with low density	• Simulation detail limited to the grid resolution • Restrictions in scalability as simulation size limited to grid size, since the fluid only exists within the confides of the grid • Limited control

Table 1.1: Collection of the comparisons between the two methods: Lagrangian and Eulerian.

PIC or FLIP method

PIC (Particle In Cell) was pioneered by the Los Alamos National Laboratory and later described by Harlow's paper Computer Physics Communications in 1962 (Strantzi, 2016). PIC was one of the early 'Hybrid' methods created to solve compressible flow problems and combined the Eulerian and Lagrangian viewpoint, where all the fluid quantities where stored on the moving particles while a grid evaluated them (Ghourab, 2011). Initially, the fluid quantities are stored on the particles and in each time step; they are transferred on the grid where they are integrated, according to the Eulerian viewpoint. Finally we interpolate the values from the grid points to the particles. The interpolation and smoothing during the particle-to-grid transfer suffered from errors and in bigger simulations from severe numerical dissipation (Bridson, 2008).

The FLIP (Fluid Implicit Particle Method) method is a simple modification of the PIC method. It was an attempt to solve the limitations of the PIC method, by Brackbill and Ruppel in 1986 (Brackbill and Ruppel, 1986). To avoid the interpolation during the particle-to-grid transfer, the FLIP method interpolates the change in the quantity (e.g. velocity) and uses it to increment the particle value, instead of replacing it. Every increment undergoes one smoothing and the result is no numerical dissipation (Bridson, 2008).

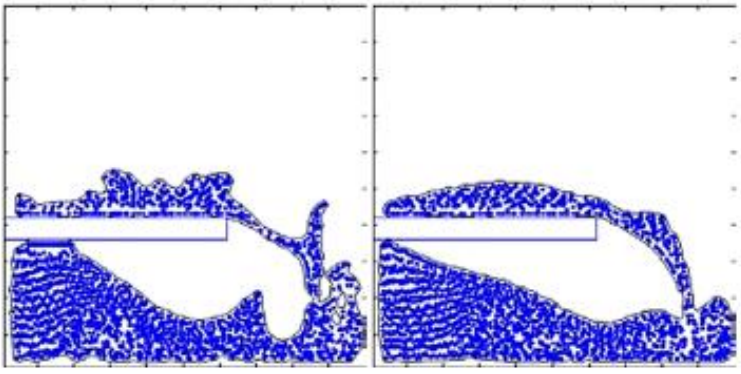

Image 1.4: A comparison of the FLIP and PIC methods: FLIP vs. PIC velocity update for the same simulation. (Image taken from Zhu and Bridson, 2005)

Hybrid Particle-Grid Methods

While the FLIP method solved the numerical dissipation problem, it may develop noise something that we don't see with the PIC method. That's why in many 'Hybrid' methods both FLIP and PIC updates are used in different blends to try and minimize the noise without introducing too much dissipation. 'Hybrid' methods were also developed to face the FLIP's limitation to first-order accuracy while nice smooth surfaces usually require higher order accuracy.

Smooth Particle Hydrodynamics - SPH

SPH is a particle-based method which implements the Lagrangian viewpoint. According to the Lagrangian method, the particles carry the fluid quantities, like velocity and acceleration but also additional attributes like density, pressure and the external forces. Initially, it was introduced by Gingold and Monaghan, and Lucy in 1977, in order to simulate astrophysical phenomena. SPH is an interpolation method which allows the replacement of the fluid with a set of disordered particles to express the equations and obtain numerical solutions (Burak, 2015). SPH's biggest disadvantage is that it can be quite unstable where the simulation does not have enough particles for neighbor calculations which results in incorrect density and negative pressure (Kuo, 2016).

2. Design

Aims and Requirements

Designing and implementing a fluid solver is a challenging process. This project focus on using the background theory and research to implement a solver in a basic simplified version with a minimum number of nodes, compared to the FLIP Fluid solver available from the shelf tools of Houdini (Image 2.1).

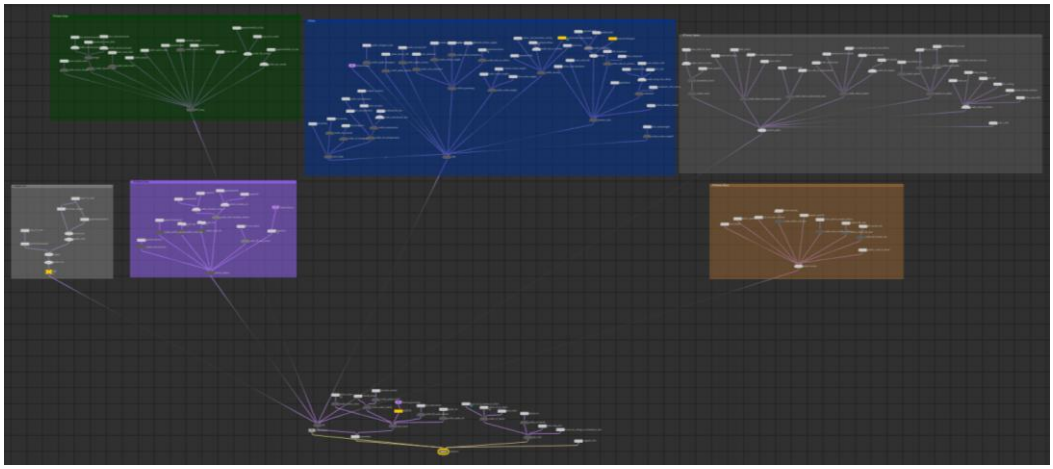

Image 2.1: FLIP Fluid Solver, built in Houdini shelf tool.

The goals were to understand the pipeline and structure of such an implementation and while making a basic solver in a very fundamental level, still achieve a visually pleasing result.

The main requirements of the framework were:

1. Create a fluid solver in Houdini from the ground up.
2. Simulating the flow of liquids like water. (Real-time visualization of fluids using particles)
3. Allowing simple interaction and static collisions with boundary containers, like boxes and rigid bodies.
4. Easily extendable and flexible to add and take away certain features in order to simulate a wide range of fluid effects.
5. Comparison between PIC and FLIP methods
6. Use solver in a number of simulations to test with different collision objects.

In order to design how to proceed with the implementation of the solver, the selected tool, Houdini had to be studied, explore its features and of course study thoroughly the already pre-built fluid shelf tools.

Houdini Software

This chapter will review the basic features and tools that the software provides with.

Every pre-existing digital asset can be unlocked, artists can dive into the network and modify its structure to support extra functionality and then save it as a new custom solver. That feature allows for more control over fluid simulations.

Since Houdini 8, node based dynamic operators (DOPS) are included and later Houdini 9 introduced fluid and gas microsolvers. A solver is actually a model that solves a set of mathematical equations that describe a phenomena in this case fluid behavior. The microsolvers will be the building blocks of the fluid solver. Below a short summary of some of the most important tools Houdini offers, is provided:

- **HDA** (Houdini Digital Asset) is a node that includes a series of operations that can be transferred between files and networks. The asset includes a network with an interface for the artist to control and manipulate a specific effect without worrying about the complexity of the process that generates it, inside the node. By wrapping up the nodes and networks of a certain effect or process into a HDA, we create a reusable tool, portable between files and networks that can be adjusted or manipulated according to the current or future needs. Digital Assets allow the generation of bigger and more complex networks by the combination of different existing HDAs making the production pipeline much faster and flexible.

- **SOP** (Geometry Surface Operators) are responsible for creating and modifying geometry.

- **POP** (Particle Operators) are used to generate the particles which can be used to model realistic natural phenomena including smoke, water and fire. (Fan, 2009)

- **DOP** (Dynamic Operators) are used to connect the geometry with forces and "configure solvers for simulating their interactions" (SideFX, Help Card).

- **Microsolvers** are nodes inside of DOPS that perform specific tasks related to the fluid solving process (Claes, 2009). Using those powerful mathematical building blocks, long

and complicated equations can be calculated and represented as a network of nodes. By wiring different micorsolvers or smaller networks of them, more complex ones can be created. The order of calculation in a microsolvers' network is left to right and top to bottom.

3. Implementation

Houdini microsolvers

After studying the Navier- Stokes equations and understanding the concepts that govern fluid motion, it is relatively easy to implement them in Houdini, since most of the mathematical equations are already implemented in the form of microsolvers. The challenge is to study the large number of microsolvers and decide which ones to use and modify them accordingly to the requirements of the desired solver.

We can solve complex mathematical equations using a combination of different microsolvers. By wiring microsolvers we can achieve bigger and more complicated networks required by the complex fluid dynamics equations, like the Navier-Stokes equations. The different microsolvers are wired together in the right order using a Merge node. The order of operations works from left to right and from top to bottom.

In the next pages, the implementation of the solver is described step by step. The process is divided in the main sections that hopefully will make it easier to understand.

Object and Forces

The first step is to create the particle system that will represent our fluid. The particles will carry the fluid quantities according to the Lagrangian viewpoint.

To achieve this we use a Points from Volume node that generates points from the volume of a given object for example a sphere. Those points will represent the particles in the fluid particle system. This provides the solver with an initial number of particles that form a specific geometry. We position the Points from Volume node in a SOP Geometry which will include all the parameters of our geometry, the particles. Its input can be any shape object; for our default solver a sphere is chosen. The Points from Volume node provides several controls with most important the Point Separation parameter which defines the minimum distance between the points. For a certain volume, the smallest the Point Separation the more particles are generated inside it. To finish up the particle system, VEX code is used to assign basic attributes to them like color, velocity, acceleration and pscale. The code is included in the Appendix (Image A.1)

To allow flexibility in the solver, after collapsing it into a sub-network, the Input 1 is added with a Switch. The user can use a Flip Object as an Input and select any object as the source of the fluid, as it is demonstrated later in the simulation examples.

Next, we add a Vector Field that will represent the 3D vector of the particles' velocity and a Vector Field Visualization Field node in order to visualize the field during the simulation. The visualization does not affect the simulation but we use it in key parts of the node structure to visualize different fields and test how they work.

The different branches of the network are connected as inputs, in the right order, to the Multiple Solver node which is responsible for integrating all the branches in order from left to right for each step of the simulation. The first input of the Multiple Solver is defining the geometry the second type of input includes all the microsolvers that operate on the geometry.

Next, we need to apply some forces to our particle system in order to animate it. A Gas Linear Combination node is used to clear the acceleration every frame and then a SOP solver is used to access the DOP geometry. With the DOP geometry as an input, a second VEX script assigns acceleration to the particles, which points in the direction of -y axis.

The forces are translated into acceleration which needs to be integrated into velocity and then velocity into position. The two integrations are calculated with the two Gas Linear Combination nodes using the ADD operator.

We multiply the acceleration by timestep so it scales dynamically in time, and we complete the integration according to the motion equation **v = v+a(t), where v** stands for the particles' velocity and **a** for their acceleration. Similarly, for the position **p**, we follow the equation **p=p+v(t).**

From this point, we start building the solver according to the FLIP requirements. We have a particle system with certain attributes: position, scale, color and velocity. The next step is to pass the particle attributes to a grid (or field), solve the necessary equations on the grid updating the attributes and pass them back to the particles.

In order to create the grids, we use the Houdini microsolvers that create and handle fields. The creation of the fields happens after we have integrated the acceleration to velocity but before the integration from velocity to position.

Fields

Velocity

Houdini has two kinds of fields: scalar and vector fields. Previously, in order to create a field to store the velocity of the particles we created a vector field named **'vel'**. A Gas Resize Field microsolver changes the size of the **'vel'** field to match the bounding box of the geometry, in this case the particles. This way, we make sure it always contains the particle system and tracks its motion.

Once the velocity field **'vel'** is set, we create an identical copy of it using the Gas Match Field microsolver, named **'oldvel'**. We are going to use that later to complete the FLIP algorithm calculations.

Surface

Gas Match Field is also used to create the surface of the fluid, which is represented by a scalar field and it uses the **'vel'** field as a reference.

Collision

In the Creating Fields section of the network, we set up another scalar field to represent the collision field adding another Gas Match Field and this time referencing the surface field.

Updating the Fields

Once the surface field is created, we can convert into a Signed Distance Field (SDF) with the help of another microsolver: the Gas Particle to SDF, which is added in the Updating Fields section of the network. This "free" surface works as the boundary condition for the particle system and we are going to reference it in a later microsolver responsible for solving the pressure equation in order to enforce the incompressibility condition of the Navier-Stokes equations.

Gas Particle to Field is copying the values of the particles' velocity into the **'vel**' field and the Gas Linear Combination node copies the **'vel**' field to **'oldvel'**, making an identical copy of the initial value of the velocity.

In the Collision Detection section of the network, during the initial stages of the solver we used
- a Gas Field VOP that uses VEX to manipulate the collision field's shape referencing the density and
- the Gas Enforcement Boundary node to make the velocity equal to zero when a collision field is present.
Those nodes were deactivated in later stages, when static objects were added as collision objects.

Incompressibility condition

Using the surface field created earlier, the pressure equation is solved by the Gas Projection Non Divergent microsolver, which removes the divergent components of the velocity field which are causing the expansion or contraction. (SideFX, help card) "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." (Agrotis, 2016)

Unfortunately, after completing all the above steps the incompressibility condition was not satisfied. While testing the liquid in various conditions, it became apparent that when the liquid was concentrated in a tank or a pool, it would shrink down to a fraction of its initial volume. An example of this phenomenon is given in Image 3.1 below. Following the theory, the problem was attempted to be solved using constant density which would create a pressure force between the particles. The pressure force would be calculated, every frame, so that the particles stay in an average distance from each other, keeping this way the density constant throughout the simulation. The attempts to calculate both the density and the pressure produced insignificant results. Due to the limited time to continue researching the problem, the focus was redirected on the strengths of the solver.

After much simulation testing, this does not seem to be an issue for running water. For that reason, the simulation examples, in the following chapter, will focus on running water phenomena.

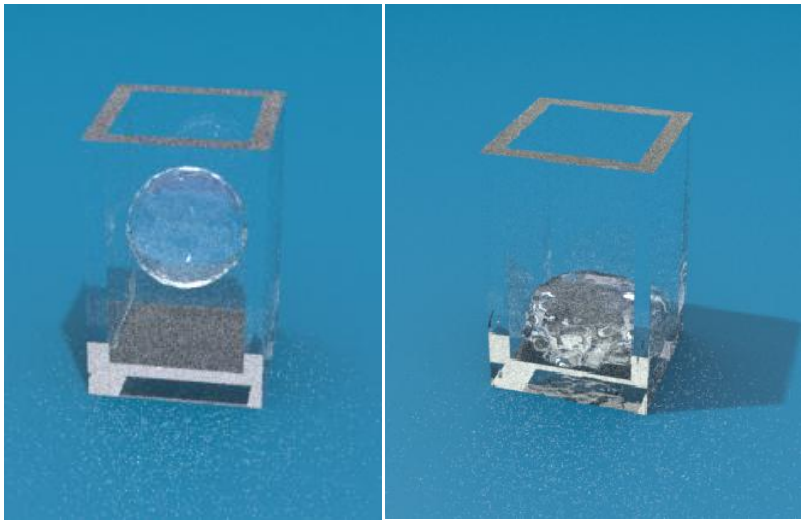

Image 3.1: Water in a glass container test.

PIC vs. FLIP

For updating the velocities back to the particles there are two available ways: the one based on the PIC method and the other on the FLIP update. Both methods were implemented and tested in the creation process of this solver before choosing the FLIP update as the default one. The reason for the FLIP selection is that allows the particles to keep the individual behavior as it is explained below.

The PIC update requires only a Gas Field to Particle microsolver to copy the '**vel**' field values to the particles' velocities. Since the particles take their velocities directly from the field where they were averaged and interpolated, their individual characteristics, like initial velocity and direction, are lost.

The FLIP update requires calculating the change in the velocity field and assigning it to the particles which keep their individual behavior. This is achieved by calculating the difference between the '**vel**' field and '**oldvel**', using the subtract operation in the Gas Linear Combination microsolver, and storing it in the '**oldvel**' field. In order to put that back to the particles, a Gas Field to Particle microsolver is used to add to the particles' velocity attribute the values of the '**oldvel**' field, which now stores the difference of the two velocity fields.

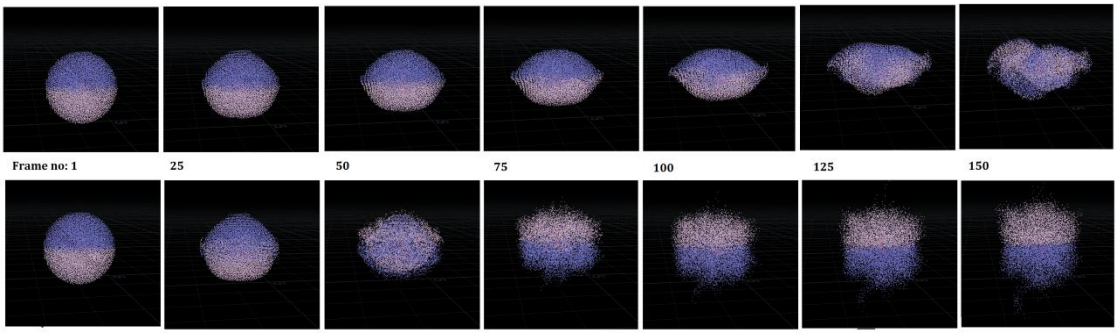

Image 3.2: FLIP updates on the top and PIC on the bottom for the same particle system. Images taken every 25 frames.

External Forces

On the default solver, the only external force that is applied to particles is gravity, which is applied directly to each particle. In later stage of the solver development, after it was collapsed into a sub-network, the Input 4 of the sub-network was used to add any additional forces to the particle. This feature was used in the third Simulation example of the spherical object were the default Houdini gravity is de-activated and a radial gravity force is added to the solver using the Input 4. The selection of that Input was made in order to match that of the FLIP Fluid Solver (shelf tool) and save Input 2 and 3 available for further development of the solver.

Collision Forces

The Collision forces are very simple and use the Static Object tool for the Collision objects in every simulation. After detecting a collision between a particle and the Static Object, the particle's velocity is reflected perpendicular to the object's surface.

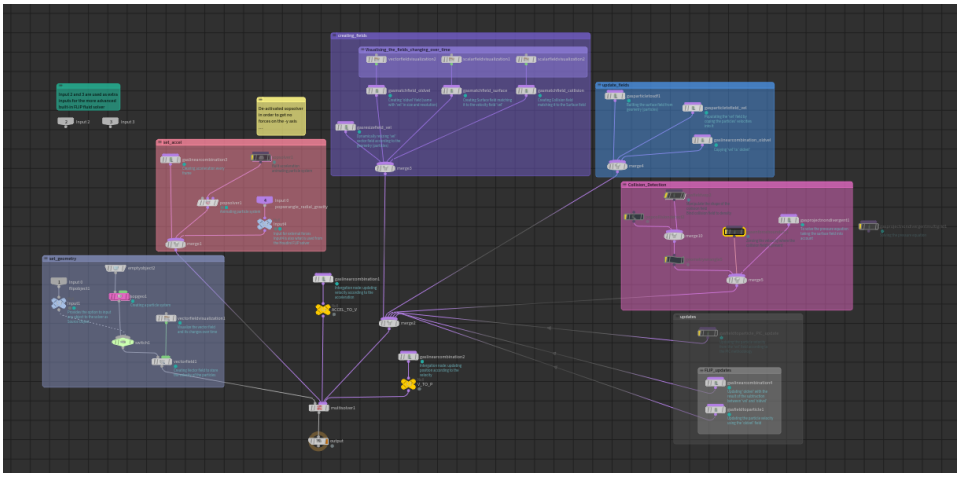

Image 3.3: The Basic FLIP Fluid Solver that was implemented in the process of the project described in this paper.

4. Simulation

Simulations examples

Using the Flip Object node as Input 1 of the solver, any shape of object can be used as source of the liquid in various scene settings.

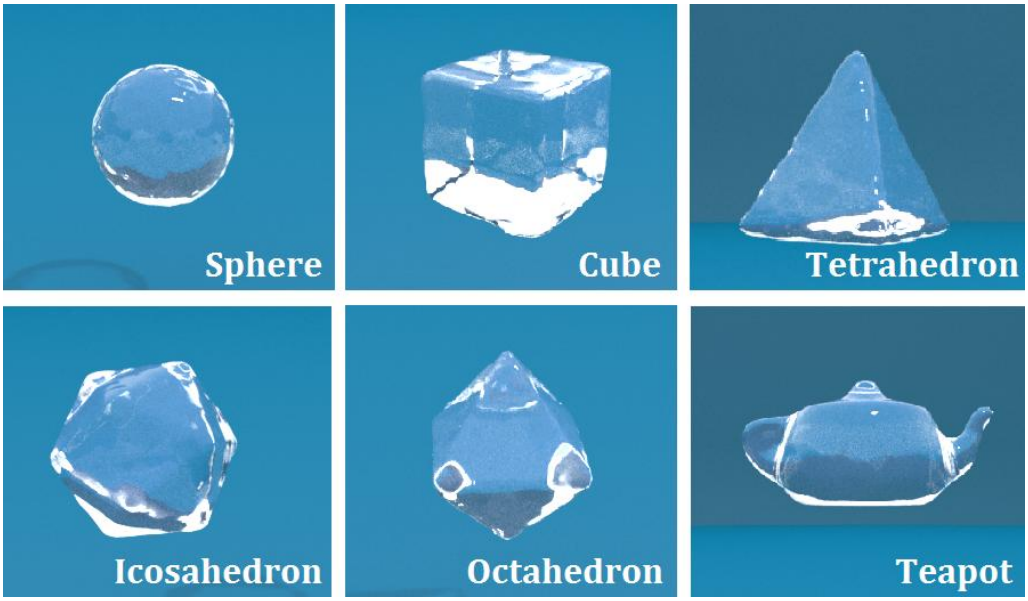

Image 4.1 : A collection of shapes that the fluid source object can take.

Floating island: Collision with columns

A scene was set up to test how the fluid particles interacts with an uneven ground, collides with objects on its path (in this case ancient temple pillars) and its the free fall due to gravity.

Simulation Parameters:
Point separation: 0.05
Particle radius scale: 1.5
Point/particle number: 320.455
Sphere uniform scale: 2.2

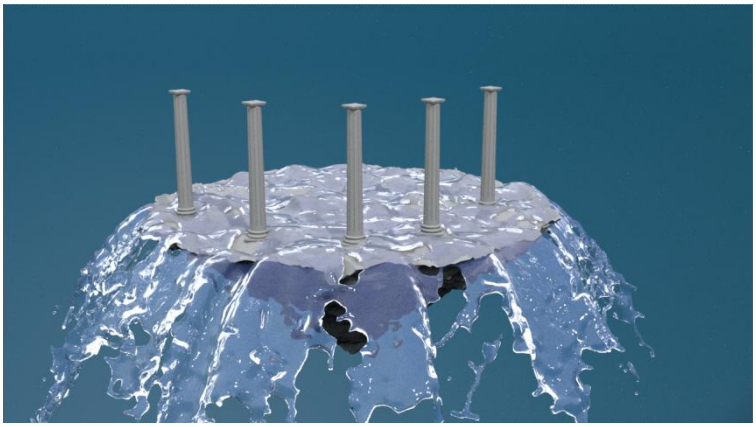

Image 4.2: Frame from the fluid simulation scene with a floating island geometry and columns as collision objects.

Floating island: Collision with wider diameter columns

The second scene is similar to the first, with a difference in the size of the pillars. Keeping all the simulation parameters the same except one, we are testing how the fluid interacts with bigger collision objects.

Simulation Parameters:
Point separation: 0.05
Particle radius scale: 1.5
Point/particle number: 320.455
Sphere uniform scale: 2.2

Image 4.3: Frame from the fluid simulation using wider diameter columns.

Spherical collision object with radial gravity

Finally, in order to test the fluid in different conditions, a scene was created were the default gravity in Houdini is de-activated and a radial gravity force is added on the solver.

Simulation Parameters:
Point separation: 0.05
Particle radius scale: 1.5
Point/particle number: 106.469
Sphere uniform scale: 1.5

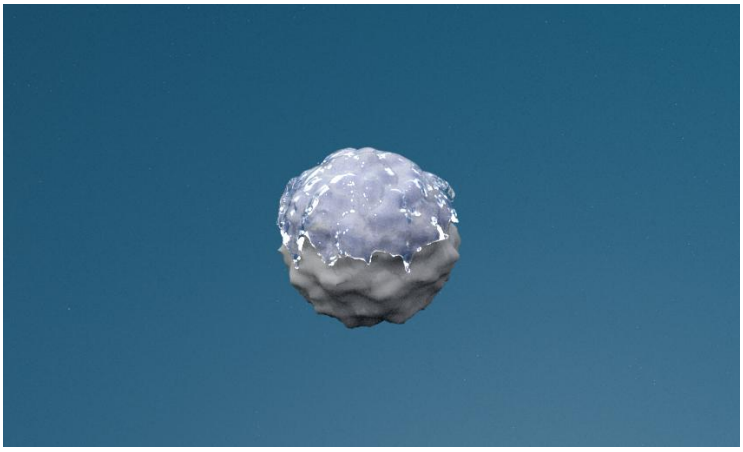

Image 4.4: Water on spherical object with radial gravity

Rendering parameters

While most of the work was dedicated on creating and implementing the fluid solver, some time had to be given to the Rendering of the simulations in order to create realistic results. The necessary lights and a camera were added in the scene and a Basic Liquid material was assigned to the surface of the liquid.

5. Evaluation

Since there was no previous knowledge of fluid simulations, a good background in knowledge and understanding had to be gained before any work began. In addition to the scientific and technical previous research review, the existing Houdini solvers and fluid tools were examined to get an understanding of how they work and how to modify them.

Results

Going back to the design stage and the goals that were set in the beginning:

1. A Basic Fluid solver was implemented in Houdini using the FLIP methodology.
2. Liquid was simulated using the solver.
3. Collision with various objects was achieved, in a good extent.
4. Added flexibility in the selection of source object and addition of external forces. Also, there is control on the number of particles generated from a specific volume (source object).
5. Comparison between PIC and FLIP methods demonstrated.
6. The solver was applied in a number of simulations testing different collision objects.

Overall, the fluid simulation is working and giving satisfying results, despite the issues that were not resolved by the time this paper was completed. By addressing those issues and with further improvements, we could get more realistic and efficient results.

The biggest issue that this project faced was that the Incompressible aspect of the fluid is not working as expected. While the incompressibility requirement was not resolved, the rest of the fluid characteristics appear to work well. The simulations with running water look very realistic. Incompressibility algorithms could be implemented in the future. Also, during the simulations, the collision was observed to show some errors on the edges. It appears to ignore objects with thin dimensions which can be seen at the edges of the floating island geometry.

Known Problems and Future Work

While most of the goals where reached, the implementation could have benefited by some more time. It covers the essential features of a fluid solver creating a satisfying simulation but they are by no means production ready. There are many more advanced aspects of this subject.

There are several improvements that could be made and problems that could be tackled in the future. The first priority would be to tackle the Incompressibility issue mentioned before as well as the collision errors at the edges of the geometry.

For future development, once the two key issues are resolved, the sub-network of the solver could be wrapped up into a Digital Asset (HDA) with a User Interface available for the artist to tweak the parameters interactively.

Conclusion

In conclusion, a design for a fluid solver was made and followed with the implementation, resulting satisfying results. Most importantly, a broad understanding of fluid simulations and solvers was acquired as well as experience using Houdini for these simulations. Because of the popularity of fluid simulations as a research topic and the fact that new developments are happening all the time, as well as the complexity of the available tools and microsolvers available in Houdini, a significant amount of time for this project went into studying and comparing methods. It was a challenging and time-demanding project that required a lot of problem solving and mathematical understanding, especially because of no previous knowledge on fluid simulations. Overall, the solvers work efficiently and the final renders are quite realistic.

Acknowledgements

Thanks to Professor Jon Macey for his support throughout the process, his advice and for pointing me to the right direction towards previous research like the projects of previous MSc students. Thanks to Professors Phil Spicer for taking the time to meet with me and discuss the progress of my project. Thanks to Michail Agoulas for his time and help.

Finally, a big thank you to my course mates for their support, precious feedback and for bouncing ideas back and forth throughout not just the project but the whole year.

References

Agrotis, Alexis, 2016. *A Fluid Implicit Particle (FLIP) Solver Built in Houdini*. Master's thesis, Bournemouth University.

Ausseloos, Mario, 2006. *Fluid Simulation.* Master's thesis, Bournemouth University.

Brackbill, J. U. and Ruppel, H. M., 1986. *FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions.* Journal of Computational Physics, 65 (2), 314–343. Available from: http://dl.acm.org/citation.cfm?id=9229 [Accessed on August 2017].

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

Burak Ertekin, 2015. *Fluid Simulation using Smooth Particle Hydrodynamics.* Master's thesis, Bournemouth University.

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

Fan, Finella, 2009. *Procedural Modelling and Animation of Breaking Waves.* Master's thesis, Bournemouth University.

Foster, N., and Metaxas, D. 1997. *Controlling fluid animation*. In Computer Graphics International, 1997. Proceedings, IEEE, 178–188.

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

Gingold, R. A. and Monaghan, J. J., 1977. *Smoothed particle hydrodynamics: theory and application to non-spherical stars.* Monthly Notices of the Royal Astronomical Society, 181 (3), 375–389. Available from: http://mnras.oxfordjournals.org/content/181/3/375.full.pdf

Gladman, Simon, 2016. *Houdini FLIP Fluid Radial Gravity.* Available from: http://flexmonkey.blogspot.co.uk/2016/11/houdini-flip-fluid-radial-gravity.html [Accessed on 28/06/2017]

Harlow, F. H. (1962). *The particle-in-cell method for numerical solution of problems in fluid dynamics* (No. LADC-5288). Los Alamos Scientific Lab., N. Mex..

Horvath, P., & Illes, D., 2007. *SPH-based fluid simulation for special effects*. In The 11th Central European Seminar on Computer Graphics, 23-25 April 2007 Budmerice, Slovakia.

Jeff Lait. *Building Fluid Solvers From Scratch,* 2012. Available from: URL: http://archive.sidefx.com/index.php?option=com_content&task=view&id=2200&Itemid=344. [Accessed on 15/06/2017]

Kuo, Rebecca, 2016. *Position Based Fluid.* Master's Project. Bournemouth University.

Lucy, L. B., 1977. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82 (12), 1013–1024. Available from: http://articles.adsabs.harvard.edu/cgi-bin/nph-

iarticle_query?1977AJ.....82.1013L&data_type=PDF_HIGH&whole_paper=YES&type=PRINTER&filetype=.pdf.

Moorhead, Jennifer, 2016. *Smooth Particle Hydrodynamics.* Master's Project. Bournemouth University.

Muller, M., Charypar, D., and Gross, M. 2003. *Particle-based Fluid simulation for interactive applications*. In *Proc. ACM SIGGRAPH/Eurographics Symp. Comp. Anim.*, 154.159.

Perseedoss. Rajiv, 2011. *Lagrangian Liquid Simulation Using SPH.* Master's Project. Bournemouth University.

Phil Spicer. H15.5_Modelling2_MSc. Houdini class material, Bournemouth University. [Accessed on 7/08/2017]

Priscott, C., 2010. *3D Lagrangian Fluid Solver using SPH approximations*. Master's Project. Bournemouth University.

Reeves, W.T., 1983. *Particle Systems - A Technique for Modeling a Class of Fuzzy Objects.* ACM Transaction of Graphics (TOG), Volume 2 Issue 2, Pages 91-108.

SideFX. Houdini Help Cards, 2017. Available at: http://www.sidefx.com/. [Accessed on 15/07/2017]

SideFX. VEX, 2016. Available at: http://www.sidefx.com/docs/houdini/vex/_index [Accessed on 15/07/2017]

Strantzi, A., 2016. Fluid Simulation Using Smoothed Particle Hydrodynamics (SPH). Master's Project. Bournemouth University.

Zhang, S., Yang, X., Wu, Z., & Liu, H., 2015. *Position-based fluid control.* In Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (pp. 61-68). ACM.

Zhu, Y. and Bridson, R. 2005. *Animating sand as a fluid*. ACM Trans. Graph. (Proc. SIGGRAPH) 24(3), 965-972.

Appendix

```
#ifndef VOP_OP
#define VOP_OP
#endif
#ifndef VOP_SOP
#define VOP_SOP
#endif

#pragma opname sop_colour
#pragma oplabel "VEX Builder SOP"
#pragma opmininputs 1
#pragma opmaxinputs 1


#include <voptype.h>
#include <voplib.h>

#include <voptype.h>
#include <voplib.h>

#include <voptype.h>
#include <voplib.h>

sop
newop_color()
{
    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);

    // Code produced by: autoconvert
    output1 = bool1;

    // Code produced by: colormix1
    if (bool1)
        clr = hsvtorgb(vop_colormix(rgbtohsv({ 1, 0.69999999999999996, 1 }),
                                    rgbtohsv({ 0.29999999999999999, 0.29999999999999999, 1 }),
                                    output1, 1));
    else
        clr = vop_colormix({ 1, 0.69999999999999996, 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 A.1: Geometry VEX code inside the SOP Geometry node.

```
#ifndef VOP_OP
#define VOP_OP
#endif
#ifndef VOP_SOP
#define VOP_SOP
#endif

#pragma opname sop_colour
#pragma oplabel "VEX Builder SOP"
#pragma opmininputs 1
#pragma opmaxinputs 1


#include <voptype.h>
#include <voplib.h>

#include <voptype.h>
#include <voplib.h>

#include <voptype.h>
#include <voplib.h>

sop
newop_acceleration()
{
    float       fval1;
    float       fval2;
    float       fval3;
    int bool1;
    vector      input1;
    vector      input2;
    vector      input3;
    vector      negated;
    vector      result;
    vector      sum;
    vector      product;

    vector      force = { 0, 0, 0};
   // Code produced by: vectofloat1
   vop_vectofloat(Cd, fval1, fval2, fval3);
   // Code produced by: compare1
   bool1 = (fval1 > 0.5);

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

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

   // Compare (>0.5)
   #ifndef
     result = (bool1 == true ? input1 :negated);
   #endif

   //Code produced by: input2 - Mass
   input2 = 1;

   //Code produced by: multiply
   product = force* input2;

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

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

   //Code produced by: bind
   accel = sum;


   // accel += result + input2;
   // printf('%v', accel);
}
```

Figure A.2: Acceleration VEX code inside the SOP solver.