A Fluid Implicit Particle Approach to a Pyro Solver in Houdini

National Centre for
Computer Animation

Ahmad Ghourab

National Centre for Computer Animation

Bournemouth University

A thesis submitted for the degree of

Masters of Science (MSc)

19 August, 2011

Abstract

Contemporary films and commercials often incorporate the use of computer generated Pyro simulations. Behind such simulations are a variety of technologies, that give artists access to a wide variety of tools, enabling them to create realistic and integrated effects. This paper looks at the implementation of a viable alternative to the current Pyro tools in Houdini, one that would allow for faster simulations, less secondary storage, slow motion functionality, and more artistic control over the flow of simulations through the use of a custom Fluid Implicit Particle Pyro solver. An extension of the solver is developed through a set of custom set-ups that allow for further artistic control, and an interaction of a variety of solvers together.

Acknowledgements

Foremost, I would also like to thank Julien Depredurand, whose guidance, advice and support was an invaluable contribution to this thesis, and a constant source of motivation. I would also like to thank Jon Macey, for all the useful feedback and suggestions throughout the course of this thesis and my masters degree. My thanks go to Dr. Hammadi Nait-Charif for the many times he welcomed me into his office to discuss my progress, and reorient my efforts to more beneficial ends. Finally, I would like to thank the entire NCCA staff and my colleagues at the MSc for their tips, support and friendship throughout the year.

Contents

List of Figures

List of Tables

1

Introduction

Most of-the-shelf software packages that feature tools allowing artists to generate fire and smoke use what is commonly termed as the "Eulerian Viewpoint" to do so. This involves the simulation of fluids on a three dimensional grid, where each cell on the grid holds information describing the current fluids quantity at that point. While this method does typically allow for realistic results, it suffers from a number of limitations, namely: speed, memory, resolution, and control.

Propitiatory pyro simulation software, developed in house by some of the largest film VFX studios, have employed the Fluid Implicit Particle (FLIP) method, implemented in this paper, to great advantage. By eliminating the memory constraints enforced by a pure Eulerian system, they are able to simulate pyro effects at much higher resolutions. By reducing CPU constraints, they are able to output more simulations in a fraction of the time. In addition, they are able to art direct their simulations with greater ease due to the implicit nature of particles.

The focus of this thesis is to create a FLIP Pyro solver in Houdini, giving artists the aforementioned capabilities, currently only accessible by artists working for large VFX studios in the form of proprietary systems.

We demonstrate the validity of our earlier criticism of pure Eulerian systems, and the success of a Houdini FLIP Pyro method in mitigating the shortcomings associated with it.

2

Previous Work

2.1 Publications

2.1.1 Harlow

The FLIP method that we implement is actually a simple modification of the Particle In Cell (PIC) method, pioneered by the Los Alamos National Laboratory in the 1950's (Harlow, 2004), and later described by Harlow's paper in 1963 (Harlow, 1963).

PIC was an early 'Hybrid' method, where particles stored all the fluid quantities, and handled their transportation, while a grid evaluated them. However, the PIC method suffered shortcoming that prevented it from accurately depicting fine fluid flow, as a result of excessive numerical dissipation (Zhu and Bridson, 2005).

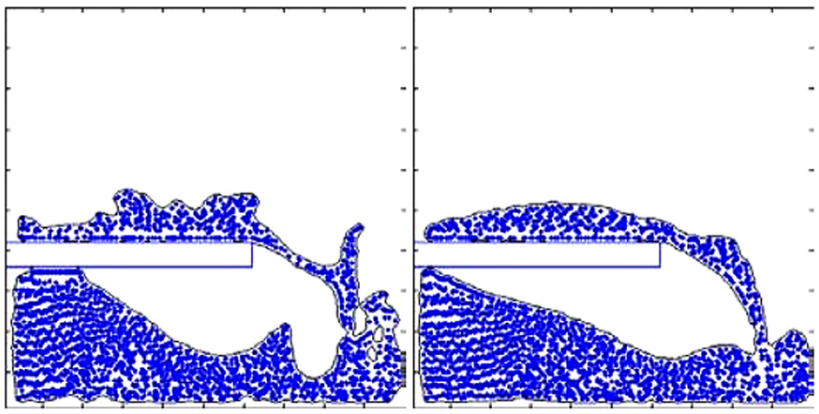

Figure 2.1: A comparison between the results of the FLIP and PIC methods for the same simulation. Image taken from Zhu and Bridson (2005).

Figure 2.2: Two Coarse Grid Step renders taken from Hovrath and Gieger (2009).

2.1.2 Brackbill and Ruppel

In 1986, Brackbill and Ruppel attempted to solve the short comings of the PIC method, by the introduction of a simple variation. This was called the Fluid Implicit Particle Method, and allowed for accurate representation of fluids, with almost no numerical dissipation (Brackbill and Ruppel, 1986).

2.1.3 Zhu and Bridson

In 2005, Zhu and Bridson introduced the FLIP method to incompressible flow. This allowed for highly detailed fluid motion on relatively coarse grids (Zhu and Bridson, 2005).

2.1.4 Horvath and Geiger

In 2009, Horvath and Geiger published a paper detailing the Coarse Grid Step, which is a two step, slight variation, of the FLIP method (Hovrath and Gieger, 2009). The Coarse Grid Step uses a simple wavelet decomposition on the velocity grid that produces multiple levels of detail. The result is then projected onto a series of two dimensional image planes that are resolved by the incompressible Navier-Sokes equations independently, producing some of he most detailed computer generated fire to date.

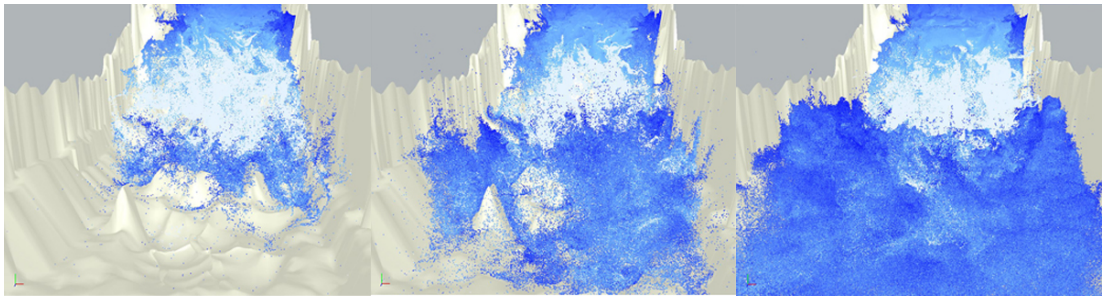

Figure 2.3: A simulation with millions of particles in the Naiad dynamics package. Image taken from (Seymour, 2010).

2.2 Off-the-shelf Systems

2.2.1 Naiad

In 2008, Nordenstam and Bridson founded Exotic Matter, developers of the Naiad fluid and dynamics engine (Nordenstam and Bridson, 2008). Naiad first saw action in the Blockbuster film *Avatar* (2009), where it was used to produce the VES award winning "drinking shot" (Seymour, 2010). Since then it has been adopted by a number of film studios as one of the primary fluid simulation tool (Nordenstam, 2011).

Naiad was praised for its capacity to produce very detailed and realistic fluid motion, while allowing for a large amount of creative control over the simulation process. The technology behind the fluid solvers use the FLIP method, or optionally in more recent releases, a variation of it known as DEFLIP. Naiad's Gas Solver is still a fully Eulerian three dimensional grid based solver (Nordenstam, 2010).

Figure 2.4: Explosion, fire and smoke simulations generated using Houdini's Pyro solver. Rendered in Mantra with the Pyro Shader. Image taken from Klosters (2009).

2.2.2 Houdini Pyro solver

In 2009, Side Effects released version 10 of Houdini (Self, 2009), which introduced the Pyro Solver, capable of producing various fire and smoke simulations. In version 11, a set of presets were created in order to enable artists to quickly achieve the desired effects. However, while Houdini's Pyro tools are capable of producing detailed realistic results, it may be possible that artists find that these tools have steep learning curve and lack of control. Furthermore, higher resolution simulations are very slow, and require huge amounts of secondary storage.

That said, the advantage of Houdini is that it is not a black box system, meaning artists can easily dive into the solver and modify the actual structure to better suit their requirements. Houdini also provides users with a rich set of solver building blocks called micro-solvers, which capable users can use to build the Pyro solvers, or their own solvers, from scratch.

Figure 2.5: A screen-shot of a detailed FLIP simulation. Image taken from (Clark, 2010).

2.2.3 Houdini FLIP solver

In 2010, Side Effects Software released the eleventh version of the Houdini 3d animation package, that also saw the introduction of a FLIP liquid solver (Self, 2010). In personal communication with Jeff Wagner, of the Side Effects support team, he noted that Houdini's FLIP solver was in fact a hack that worked "O.K.".

The solver is capable of handling hundreds of thousands of particles to produce complex fluid motion with a fairly coarse grid, and requires no sub-stepping. Furthermore, the solver is also unconditionally stable, making it advantageous over the previous Houdini Volume and Smooth Particle Hydrodynamic solvers in many situations (Lait, 2010).

2.3 Proprietary Systems

2.3.1 ILM Plume

We have previously discussed the predecessor to Industrial Light and Magic's Plume dynamic simulation engine and renderer (Failes, 2010), namely the publication by Horvath and Gieger (Hovrath and Gieger, 2009). Plume is only a slight variation of their method, as they abandoned the 2D slicing of the Coarse Grid Step refinement process, opting for a fully 3d refinement step instead, simulated on the GPU (Desowitz, 2010). This was used with great success in creating the fire simulations seen in the film *The Last Airbender* (2010).

2.3.2 DNeg Squirt

Double Negatives Squirt is another proprietary fluid simulation package that uses the FLIP method, simulated on the GPU, to create very detailed high resolution fluid simulations. Originally developed by Exotic Matter founders Marcus Nordenstam and Dr. Robert Bridson (Seymour, 2010).

Figure 2.6: An example of the level of detail achieved with ILM's Plume 3D Coarse Grid Step system in *The Last Airbender* (2010). Image taken from (Failes, 2010).

Figure 2.7: Pyroclastic ash clouds from the film *2012* (2009), made using Double Negative's squirt dynamics engine. Image taken from (Desowitz, 2009).

3

Theory : Fluid Implicit Particle Method

3.1 Navier Stokes

3.1.1 The incompressible Navier Stokes equations

The Incompressible Navier-Stokes equations is composed of a set of partial differential equations that form an excellent model to simulate fluid flow (Stam, 1999), commonly used as the basic building block of many solvers.

Strictly speaking, one need not understand the maths behind the Navier Stokes equations in order to implement a fluid solver of their own within the Houdini framework. Instead, only an understanding of the concepts that govern fluid motion, as set out by the equations, is necessary, as most of the maths is already implemented and ready for use in the form of microsolvers. A more in-depth explanation of the Incompressible Navier-Stokes equations can be found in Bridson (2008).

The Navier-Stokes equations usually appear in the form of:

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

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

Where:

$\vec{u}$	velocity
t	time
ρ	pressure
$\vec{g}$	external forces
ν	viscosity

Equation (3.1) is called the momentum equation, and is actually quite simply a variation on Newton's second law of motion $\vec{F} = m\vec{a}$ (Bridson, 2008, pg. 4). Equation (3.2) is the incompressibility condition.

3.1.2 Advection

$(\vec{u} \cdot \nabla)\vec{u}$

Advection is defined as the transport of matter by the flow of a fluid (Bridson, 2008, pg. 8). Visually we can think of this as the rising smoke from a lit cigarette, where chemical reactions at the end of the cigarette release, amongst other things, soot particles and heat. In this case, heat flowing to cooler areas of the surrounding environment carry the soot particles along with it. The horizontal and vertical components of this motion are referred to as advection.

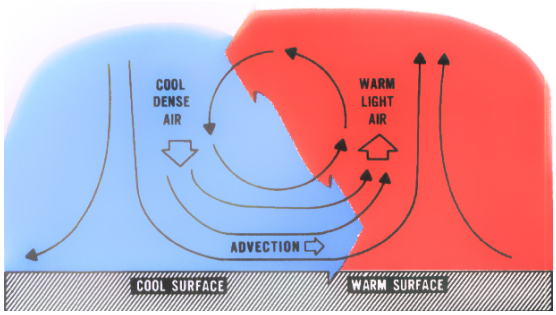

Figure 3.1: An image of depicting atmospheric advection. Image taken from F.A.A. (1975).

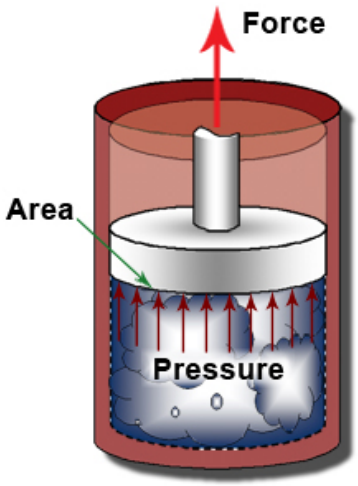

Figure 3.2: Pressure = Force per unit of area. Lowering the piston increases pressure, and vice versa. Image taken from Baratuci (2006).

3.1.3 Pressure

p

Pressure is defined as the force per unit area(Harris, 2005, Chap. 38). Higher pressure areas will have an outward motion, exerting force on lower pressure regions. This can be visualised as a gun being fired, where the initial burning of gunpowder causes a significant release of hot gasses in the confined space of a gun's chamber, resulting in a very large force per unit area, i.e. pressure, that propels the bullet out of the gun's barrel.

3.1.4 External Forces

$\vec{g}$

This term, in the context of the Navier Stokes equations, refers to the combination of all external forces that act on a body of fluid(Harris, 2005, Chap. 38). There are two main classifications of external forces, local forces or body forces. Local forces are any force that results in only a part of the fluid body to accelerate, such as buoyancy. External forces are any force that result in the entirety of the fluid to accelerate, such as gravity.

Figure 3.3: An image of several liquids with different levels of viscosity. Image taken from (Rutter, 2011).

3.1.5 Diffusion

$\nu \nabla \cdot \nabla \vec{u}$

Viscosity is defined as the measure of a fluids resistance to deformation and flow(Harris, 2005, Chap. 38). The more viscous a fluid is, the slower and more uniform its direction of motion along a surface. Visually this may be the difference between water and honey. Water has very low viscosity, it changes form and flows with ease, the direction of its motion uninhibited. Honey on the other hand has a high viscosity, its shape relatively uniform, and its flow along a surface cumbersome.

3.1.6 The Incompressibility Condition

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

In nature, the volume of fluids can change, but rarely significantly enough that a visual difference is noted. Simulating the change in volume of fluids is complicated and computationally expensive, as such we presume that our fluids are incompressible and that the volume is maintained throughout the simulation.

3.2 Fluid Viewpoints

There currently exist three main viewpoints to simulating three dimensional fluids in computer graphics. A brief introduction to the methods is necessary in order to better understand the motivation for the use of a Fluid Implicit Particle method to simulate our pyro effects.

3.2.1 Eulerian Viewpoint

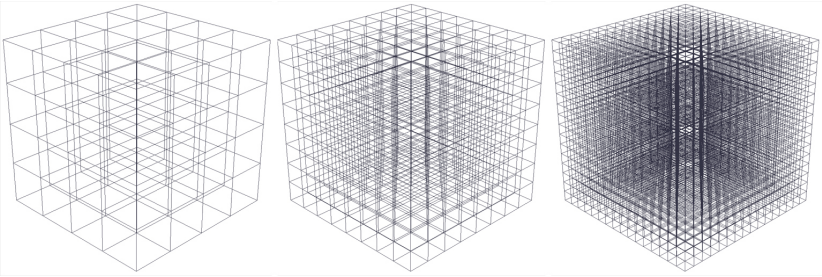

Figure 3.4: Three Eulerian grids, with resolutions of 4^3, 8^3 and 16^3 respectively. Note that doubling the resolution increases details eight fold each time.

With the Eulerian viewpoint, named after the famous Swiss mathematician Leonhard Euler, we concern ourselves with a specific discretized finite volume of space. This is typically represented as a three or two dimensional grid, where fluid quantities are stored, evaluated and flow by each individual cell of the grid (Bridson, 2008, pg. 7). In a Eulerian system, the fluids may only exist within the confines of the grid, and as such they are effectively bounded. This limits the simulation of the fluids to a particular location.
Visualization : Weather forecast station.

3.2.2 Langrangian Viewpoint

With the Langrangian method, named after the French mathematician Lagrange, we deal with a set of individual particles that carry with them fluid quantities, in addition to having their own unique position and velocity(Bridson, 2008, pg. 6). The particles are unbounded, and as such are capable of occupying an infinite space.
Visualization : Weather balloon.

3.2.3 Hybrid Viewpoint

In a hybrid method, components of both the Eulerian and Langrangian methods are used to advantage. Fluid quantities are carried by particle masses that flow through an Eulerian grid. Advection is handled by particles, and all other fluid quantities are integrated on the grid. At every time step, grid values are set as a weighted average of nearby particles, and upon evaluation, the particle values are interpolated from the updated grid values(Bridson, 2008, pg. 137).
As with the Eulerian method, the system is bounded and may occupy a finite space.

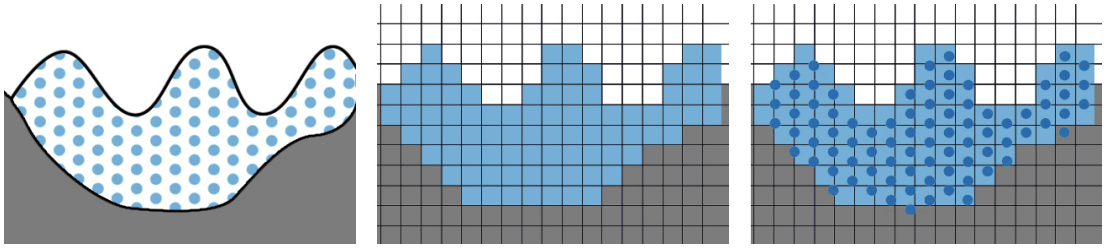

Figure 3.5: A Langrarian, Eulerian and Hybrid viewpoint respectively.

3.2.4 Advantages and Disadvantages

Each viewpoint can be implemented in a number of ways, however, as we are primarily concerned with fluids in the Houdini context, we will compare the current implementation of each viewpoint within Houdini.

Feature	Volume - Eulerian	FLIP - Hybrid	Smooth Particle Hydrodynamics - Langrangian
Substeps	10x	1x	100x
Data Storage	Volumes	Particles	Particles
Control	Fields	Full	Force Velocity
Distribution	Even Slices High Speed	Arbitrary Slices High Speed	Arbitrary Slices Good
Problems	Viscosity Evaporation Memory Intensive Grid Limited Resolution Grid Limited Size Lacks Slow-Motion	Compression Large Substeps	Explosions Slow
Advantages	Performance Independent of Particles	Fast Memory Efficient Coarse Grids Easier Manipulation with Particle Tools	Unbounded Easier Manipulation with Particle Tools

Table 3.1: Houdini fluid methods comparison table. Modified from Lait (2010), with additions from Priscott (2010, pg. 4) and Bridson (2008, pg. 6).

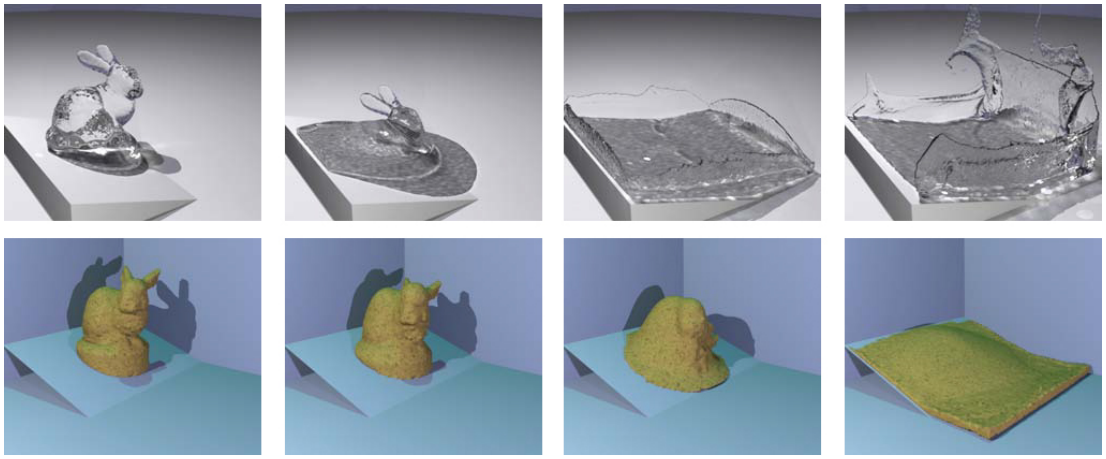

Figure 3.6: Simulations of sand and water using the FLIP method, taken from Zhu and Bridson (2005).

3.3 Fluid Implicit Particle Method

3.3.1 Particle In Cell

The Particle In Cell (PIC) method is an early hybrid method developed by Harlow (Harlow, 1963). As defined in 3.2.3, particles carry the fluid quantities and a grid integrates them. Advection is handled by the particles.

The problem with PIC is the excess of numerical diffusion as a result of repeatedly averaging the weight of particles onto grid cells, then re-interpolating them back onto the particles. As a result of the numerical diffusion, fine details are lost, making this an unsuitable method to simulate fluid motion with an acceptable level of detail(Zhu and Bridson, 2005).

3.3.2 Fluid Implicit Particle

In 1986, Brackbill and Ruppel published a paper on the Fluid Implicit Particle (FLIP) method that remedied the aforementioned issues with the PIC method. Unlike the PIC method, in the FLIP method particle values are updated by the difference of the evaluated grid values, from the start of the time-step, as opposed to being completely overwritten by the new grid values.(Brackbill and Ruppel, 1986).

3.3.3 Solve Steps

The steps of the FLIP method, as defined by Zhu and Bridson (2005) are (modified for a Gas Solver):

1: Initialize particle positions and velocities.
For each time step:

[2] At each cell in the grid, compute a weighted average of nearby particle fluid quantities.

[3] Save the grid fluid quantities.

[4] Do all non-advection steps of a standard pyro solver.

[5] Subtract the new grid fluid quantities from the saved fluid quantities, then add a percentage (FLIP - typically 95%) of the difference, and a percentage of the grid value (PIC - typically 5%), to each particle.

[6] Move particles through the grid velocity field with an ODE solver, making sure to push them outside of solid wall boundaries.

[7] Write the particle positions to output

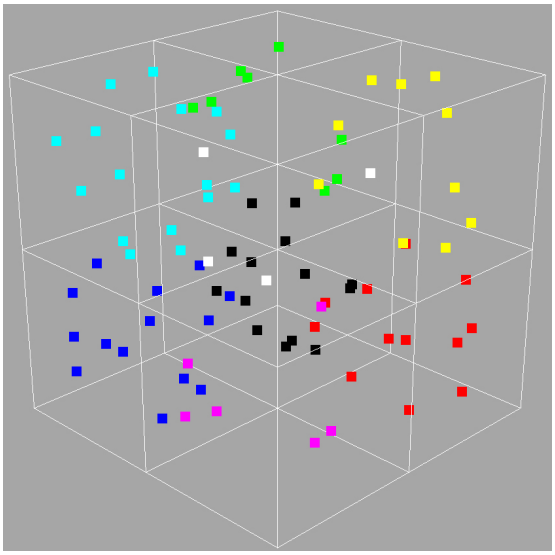

Figure 3.7: Between 4 and 16 particles are seeded in each grid cell to prevent gaps or noise in the simulation.

3.3.4 Initialise Particles

Typically 8 particles are seeded per cell (Bridson, 2008, pg. 147), and are randomly jittered to avoid aliasing when our flow is under-resolved at the simulation resolution (Zhu and Bridson, 2005). It was found that seeding significantly more or less particles would allow for too much noise or occasional gaps in the fluid flow respectively (Zhu and Bridson, 2005). During implementation, it was noted that while seeding too few particles did indeed result in gaps, seeding significantly more particles did not always introduce a significant increase in noise, and was at times desirable.

3.3.5 Transferring to the Grid

At every time step, we copy a weighted average of the nearby particles onto our grids. Convention dictates that nearby particles are defined as those who exist in a cube that is twice the grid cell width (Zhu and Bridson, 2005). During implementation, it was found that this need not necessarily be a rule of thumb, and that finer details could be achieved by reducing the extrapolation distance. Increasing the extrapolation distance further produced more cohesive results. Manipulating the extrapolation distance allows for artists to attain further control over the movement of the fluids. Furthermore, it was

also found that when simulating on higher density grids, increasing the extrapolation distance was a necessity.

3.3.6 Updating Particle Velocities

At every time step, we trilinearly interpolate the difference in the fluid quantities, and a fraction of the grid value, back to our particles(Zhu and Bridson, 2005). This is effectively a combination of the PIC and pure FLIP methods.

If we were to only copy the difference (i.e. purely FLIP) back onto the particles, the system would develop a significant degree of noise as there is no cohesion between the particle what so ever(Bridson, 2008, pg. 149). By adding a fraction of each system, typically 95% FLIP and 5% PIC, to the particle, we eliminate the noise while not introducing significant dissipation.

By varying the ratio of FLIP to PIC in our system we can effectively control viscosity as an added bonus. During development, a ratio of 70% FLIP and 30% PIC was found to produce results that most closely resemble the original Eulerian solver.

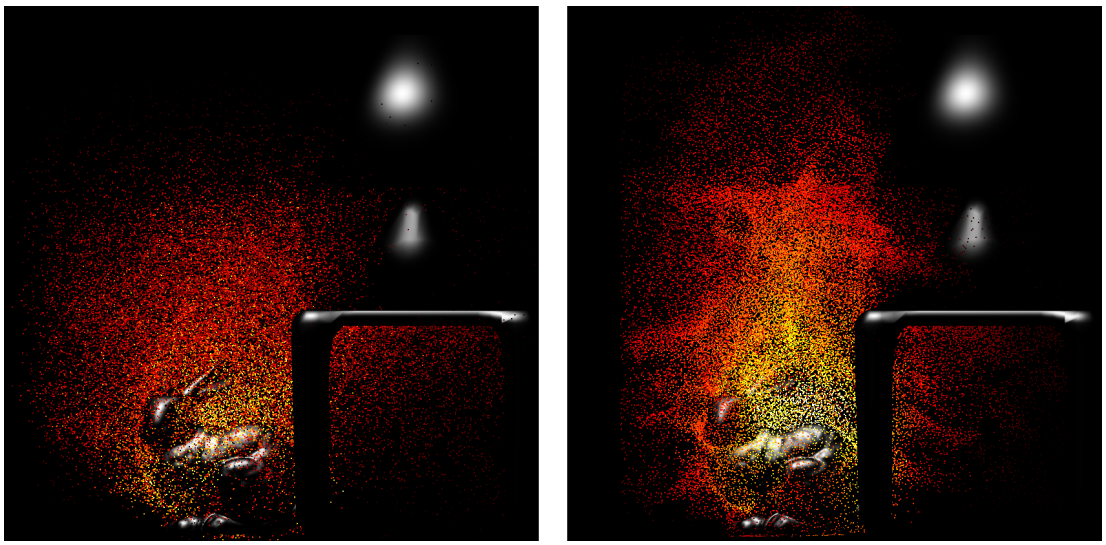

Figure 3.8: An example of pure FLIP noise. The image on the left has a FLIP to PIC ratio of %100 to %00, whereas the image on the right has a ratio of %80 to %20

4

Motivation

There exist a number of reasons why a FLIP Pyro solver would be considered advantageous to that of a purely Eulerian grid based method. In this chapter we discuss the advantages and pitfalls of the FLIP method to that of the Houdini Volume Eulerian method.

4.1 Speed

In a FLIP method, multiple particles, convention dictating around 8 for reasons discussed in 3.3.4, are seeded in each cell. Each particle may be seeded with, or incremented from interpolation, unique fluid quantities and velocity. This allows for many more degrees of freedom in the particles than in the grid(Bridson, 2008, pg. 149).

This produces a significant speed advantage, as we only need to solve our fluids on a grid that is a fraction of the resolution of a purely Eularian grid to achieve the same level of detail (Hovrath and Gieger, 2009).

When simulating both methods with the same Grid resolution, ignoring the extra degrees of freedom of the particles, the Eulerian method is marginally faster, as a result of the absence of the interpolation of fluid quantities to and from the particles process.

4.2 Resolution

The FLIP method usually requires a significantly lower resolution grid to achieve the same level of detail in a fluids motion by comparison to that of a purely Eulerian

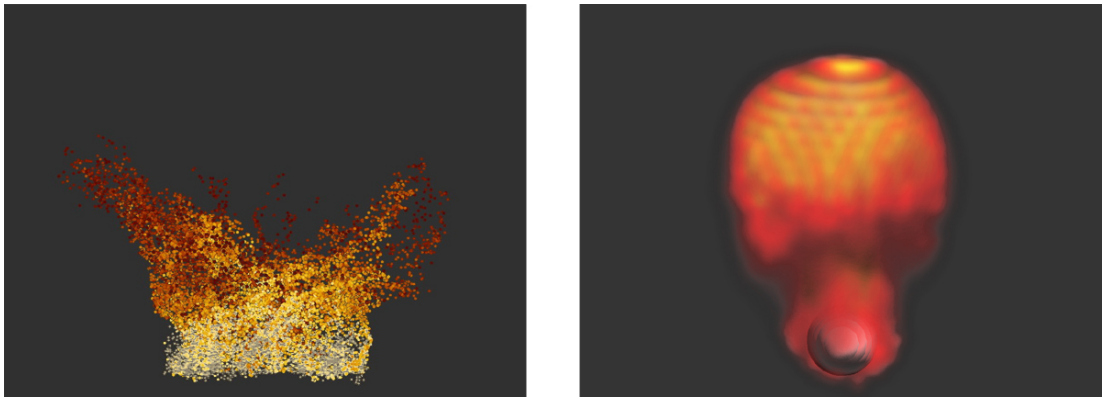

Figure 4.1: The image on the left, taken from Hovrath and Gieger (2009), was simulated using the first stage (FLIP) of the Coarse Grid Step method, on a grid with 50^3 grid cells. The image on the right was simulated in Houdini with the same number of cells using the Volume method. The former contains significantly more details despite having the same number of grid divisions.

method (Hovrath and Gieger, 2009). During Implementation, it was found that particles simulated on a grid with a spacing of 0.1 units could continually add fine details to grids with a spacing of 0.01 units, effectively 10 times level of detail of the original simulation.

4.3 Slow Motion

Simulating a fully Eulerian grid based fluid at a very high frame rate results in significant smoothing and general loss of detail. This is a result of excessive numerical dissipation produced by semi-Lagrangian advection(Zhu and Bridson, 2005, pg. 35). At every time step, when using the semi-Langrangian advection scheme, we take a weighted average of values from the previous time step, and as such are doing an averaging operation (Bridson et al., 2006, pg. 35). When we increase the number of time steps we are effectively increasing the number of averaging operations, and thus smoothing results and loosing details.

In a Fluid Implicit Particle method, advection is handled by the particles. As we are only incrementing the change of fluid quantities there is a negligible dissipation.

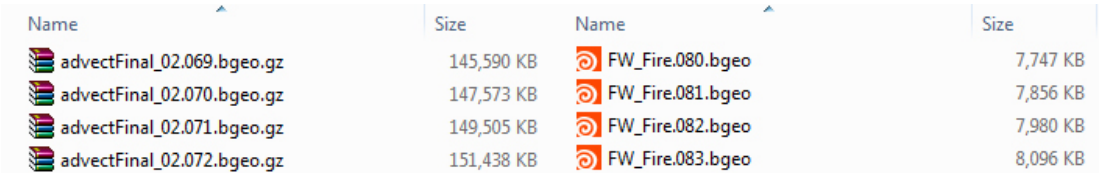

Figure 4.2: The files on the left were generated by a Houdini Pyro Volume solver with 380^3 grid cells (Ghourab, 2011), and were saved in a compressed format. The files on the right were generated from our 'Fire + Water' example, which at render-time was converted into a volume with approximately 400^3 grid cells at its peak scale.

4.4 Memory

Eularian three dimensional grid based methods can be very expensive. Doubling the resolution of the grid effectively requires 8 times the memory and roughly 8 times the calculation times (Bridson et al., 2006, pg. 63). Furthermore, most grid methods require us to store grid cells that may have no fluid quantities stored within them or any bearing on the simulation.

With the FLIP method, there are no persistent fields. All fields are created at the start of every time step, and destroyed at the end of every time step, and as such we need not store any grid data on secondary storage(Zhu and Bridson, 2005). Particle masses are instead saved. This is advantageous as the particles typically have a smaller memory footprint than three dimensional grids. Furthermore, we need only store particles that have significant bearing on the simulation. Any particles that stray from the simulation, or no longer make any significant contribution to the simulation can be trivially removed.

4.5 Control

In an Eulerian system, we may add custom controls, such as wind and gravity, to better refine the fluid motion to suit our needs. These forces are then combined and introduced into the system as a single force, denoted as $\vec{g}$ in the Navier-Stokes equation.

As the FLIP method is inherently particle based, this allows us to subject the particles in the simulation to traditional particle manipulation techniques, in addition to the aforementioned field forces.

5

Houdini Technical Background

5.1 Introduction - Dynamics System (DOPS)

The Houdini DOP context handles all non-particle dynamic simulations within Houdini. Houdini's DOP context behaves differently to other contexts. Data flows through a network in the order of top to bottom, left to right, and the state of frame is stored and updated from at the next frame. Furthermore, most data is hidden from the user, visible via Houdini's details view.

In this chapter, we unravel Houdini's DOPs system, and have a closer look at the building blocks of Houdini's solvers, its *micro-solvers*.

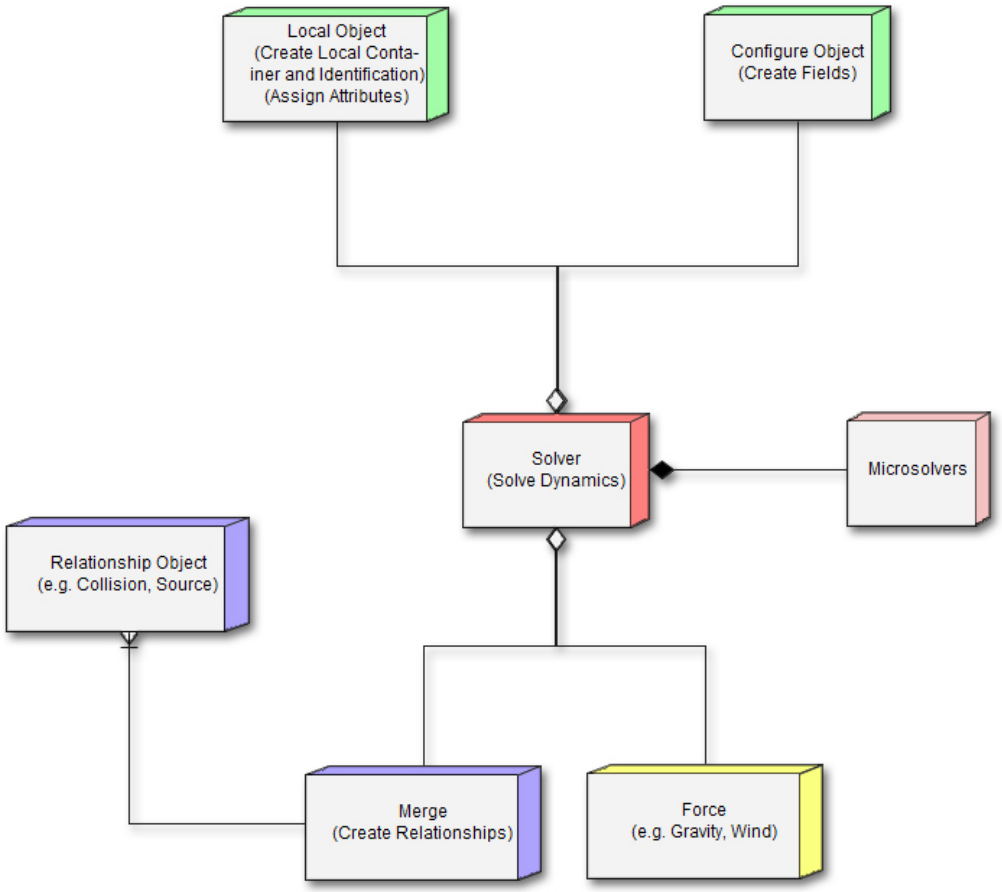

Figure 5.1: An overview of the DOP Network necessary for our FLIP Pyro solver.

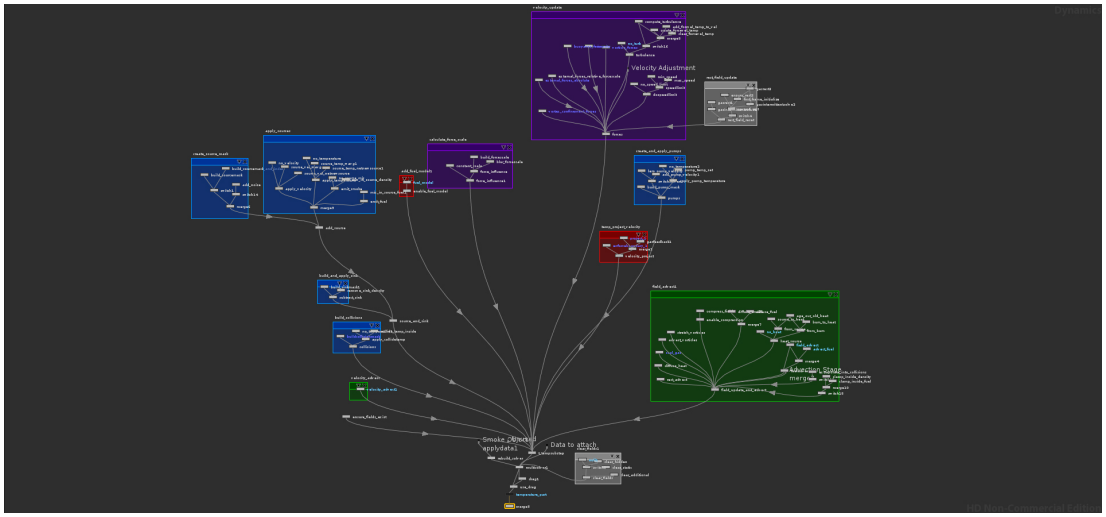

Figure 5.2: A screen-shot of the Volume Pyro Solver's interior network in Houdini, which any user may modify or extend.

5.2 Houdini's Open System

One of the most important aspects of Houdini's DOP system is that it is 'open', that is to say it is not a black box system. The user is given access to the networks contained within most DOP nodes that perform a number of functions. An example of this is fluid solvers, where a user may dive into the solvers node tree and modify its structure. The user may also create a new custom solver of his own, as all the necessary functions to do so are provided to the user in the form of micro-solvers and DOP nodes.

5.3 Creating a local object

The first step to creating a simulation within DOPs is to create an local dynamics object that represents the type of data we wish to simulate and its initial state. All objects in DOPs must have a unique DOPs identification and must be contained within a Houdini dynamics container, as Houdini's DOP context cannot directly handle external data. To do so, we lay down the first node in our network, an Empty Object DOP. This node simply creates an empty container with a unique name that the DOPs context can see and manipulate.

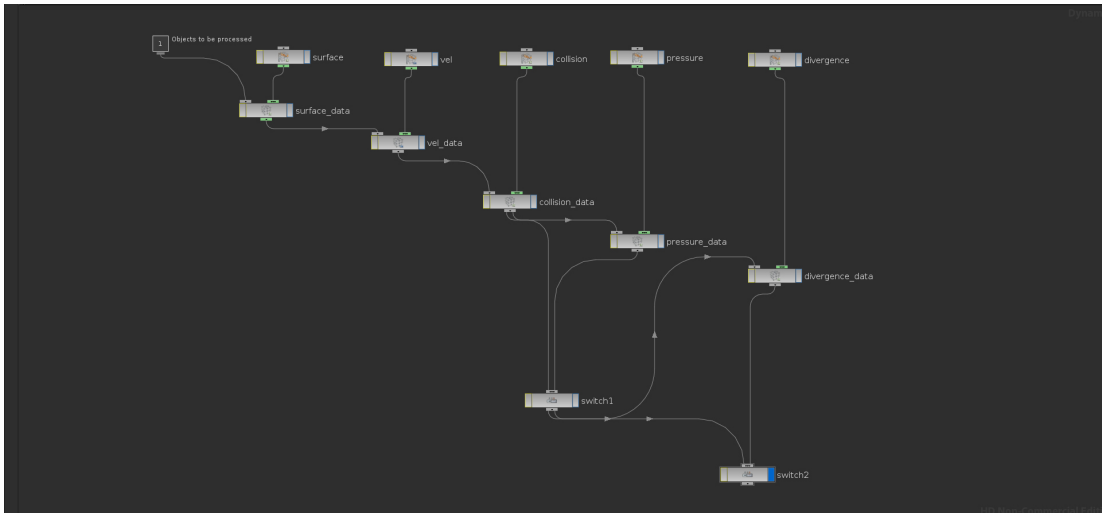

Figure 5.3: A screen-shot of the FLIP Liquid configure objects interior network in Houdini. Data is piped in via the top-left most node. Nodes at the top are field visualisers, and nodes at the bottom are our scalar and vector fields to be visualised.

5.4 Configure Object

However, it's still empty, so our next step is to fill it up with the relevant data. There are approximately two main data types, geometry, which may be the mesh of a Rigid Body Dynamics (RBD) object, and fields, which are Eulerian three or two dimensional grids that may contain scalar or vector data.

This is done using a *configure object*, which allows us to import external data into our container, set its initial attributes and create persistent fields. For geometry we may attach physical attributes such as mass, bounce or friction. For fluids we may add fields that store its density, temperature or pressure.

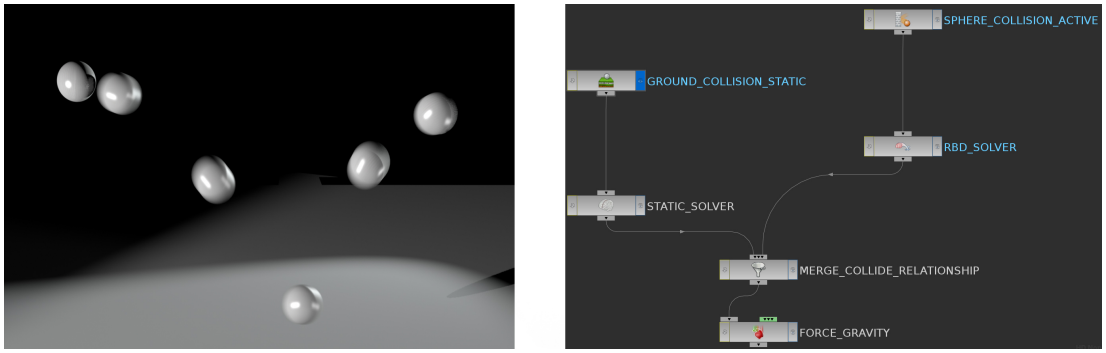

Figure 5.4: A RBD ball bounding on a ground surface. The merge node creates the collision relationship between the ball and the ground objects.

5.5 Forces

As we know from Isaac Newton's apple analogy, the apple only falls to earth as a result of gravity, and in the absence of it would remain stationary and attached to the tree. The same is true for our Dynamics objects, as simply having physical properties will not translate into anything interesting in the absence of forces. Houdini gives users access to a number of default force nodes, such as gravity, wind and vortex. However, for good measure Houdini also allows users to create custom forces using a VOP Force node.

5.6 Merge Objects

Most dynamic simulations usually involve a large array of objects interacting with one another. Visualise water gushing from a broken fire hydrant, breaking against the overhead shop curtain, its cloth swaying wildly, the ground beneath absorbing falling water droplets.

In order to correctly simulate all the interactions, Houdini provides us with the Merge DOP, which merges several objects into the same stream and creates affector relationships between the various streams. It allows us to cause the water to collide with the overhead shop curtain using the *collision* relationship, and allows us to delete water that collides with the ground using the *sink* relationship. There are several more relationships it allows us to use in order to bring scenes to life and create believable interactions.

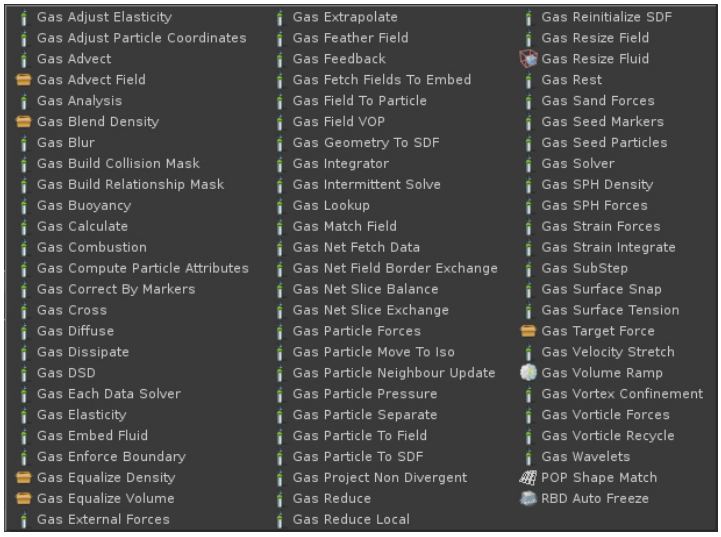

Figure 5.5: A screen-shot of the solver building blocks, *microsolvers*, available in Houdini 11.

5.7 Solvers

The Solver role is to take all the attached objects and integrate them together. Although we set our initial objects attributes, connect a force to, perhaps even assign collision geometry, there is nothing to evaluate them interacting together.

Solvers integrate them together, allowing objects to behave in a physically believable manner, for forces to manipulate them correctly, and for collision objects to affect their path of motion accurately. The solver can be thought of as the brain of the system.

5.8 Micro-Solvers

Micro-solvers are the basic building blocks of Houdini's solvers. Houdini has a total of 74 microsolvers, which in combination together can create any type of solver. Furthermore, for volume data, if a user finds that there exists no custom micro-solver suitable for his purposes, he may make one of his own using the Gas Field VOP node.

6

Implementation

Removed from public version.

7

Custom Setups

7.1 Slow Motion

As mentioned in 4.3, Eulerian based solvers are incapable of creating ultra slow-motion simulations. Our FLIP pyro solver on the other hand is capable of slow motion simulations by default. This is due to the fact that our fluid quantities are carried by the particles, and their values are unaffected by the advection process.

In our setup, we simulate the muzzle flash from a discharging gun. Our fuel and source object is a sphere that encompasses the last fraction of the guns barrel, and a fraction outside of it. As objects are heated up, and gas is released, this causes a forward and outward motion of our fire. A large quantity of particles are seeded at the initial frame, which makes up the bulk of the muzzle flash, and a small number are seeded thereafter to create an after flame on the tip of our barrel.

To enable slow-motion we simply change our scenes frames-per-second attribute to match the duration we would like our simulation to take. In our case, we adjusted our scene to 180FPS, which slows the simulation down by a factor of 7.5.

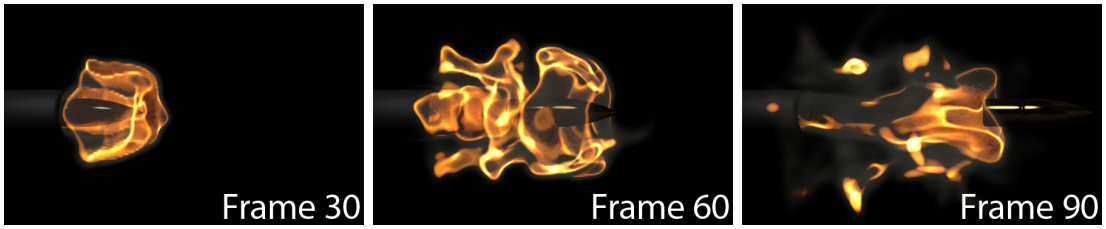

Figure 7.1: Our muzzle flash, slowed down by a factor of 7.5x (180fps).

7.2 Solver Networking - Fire + Water

We mentioned earlier the use of Merge DOPs to create relationships between objects on the DOP level, however this does not give us the necessary functionality to allow for two or more solvers to affect one another based on attributes present in any of the interacting systems. For example, we know that in nature, water may be used to quench a fire. In order for a similar effect to be simulated within Houdini we require a custom extension of our solvers. This section details a custom work-flow that allows for the interaction of other particle based solvers with our FLIP Pyro solver, and demonstrates its validity by causing a FLIP liquid solver to collide with, advect and put out a separate FLIP Pyro system.

Figure 7.2: Liquid colliding, advecting and cooling our FLIP Pyro fire.

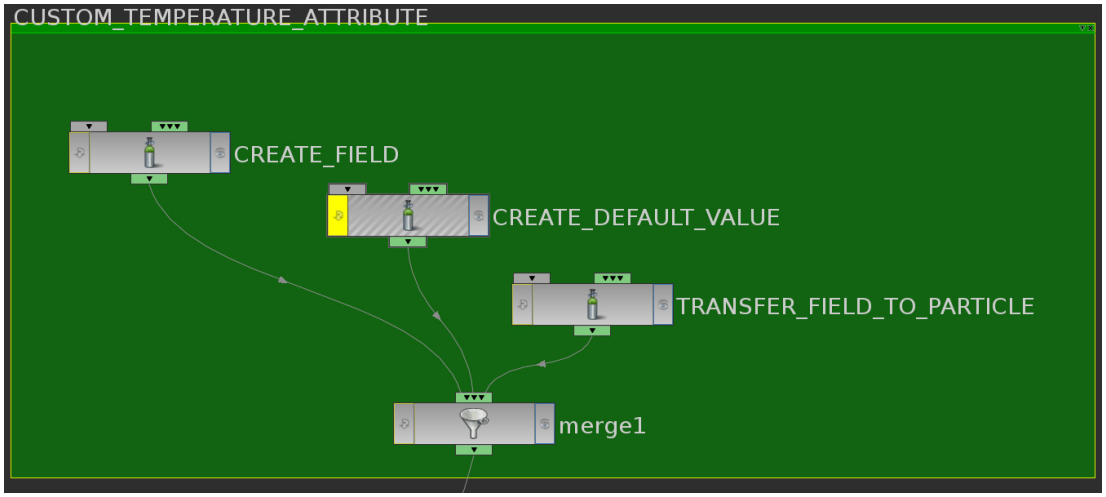

Figure 7.3: We create and initialise FLIP liquid temperature attribute.

7.2.1 Custom Fields

The first issue we may run into when attempting to influence our solver using any of the existing solvers is the lack of standardised attributes on all the independent solvers. The FLIP liquid solver, for example, lacks a temperature attribute. Houdini allows an end user to add custom fields and values to any of the existing solvers without the need to modify the contents of the solver itself.
Giving our FLIP liquid solver a temperature attribute is actually fairly simple:

For each time-step:

1: Create temperature field using Geometry as reference for scale.

2: Set the initial value of our temperature field to zero.

3: Copy our field values onto the particles.

4: Feed newly created nodes into the third input of our FLIP liquid solver, ensuring data gets attached.

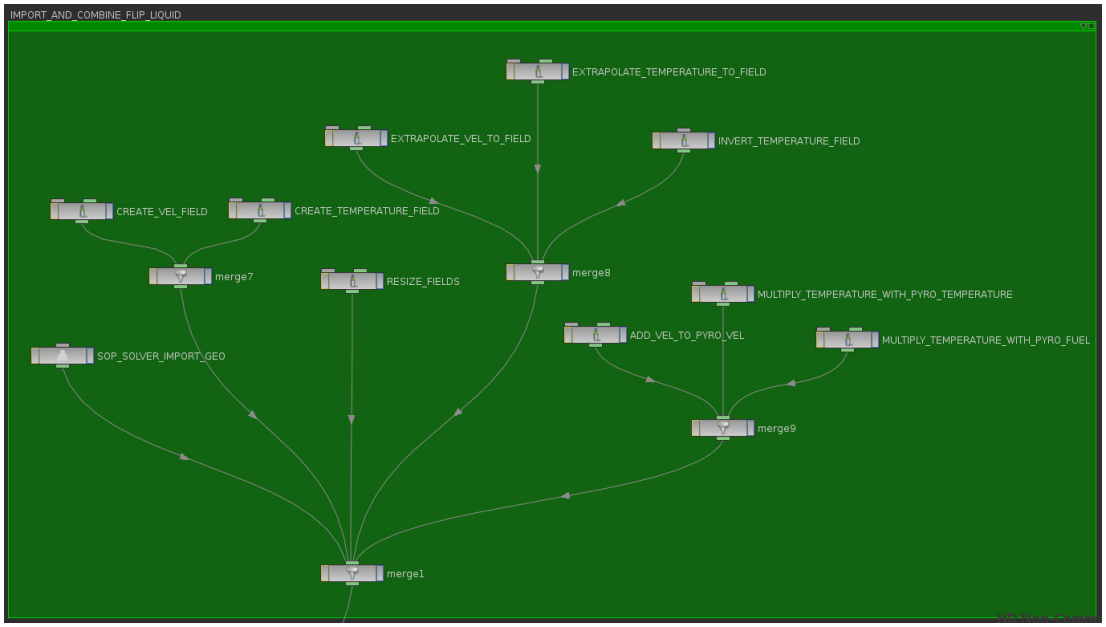

Figure 7.4: Our advection and negation setup.

7.2.2 Advect FLIP Pyro

Using a SOP Solver DOP, we can import our particle Geometry at every frame into our FLIP Pyro network. To advect our Pyro effect, we need to combine the velocity of both our simulations, and feed the result back into the FLIP Pyro solver. The steps to do this are:

For each time-step:

1: Import FLIP Liquid particle geometry, assign local unique identification.

2: Create temporary velocity vector field.

3: Resize temporary field to match FLIP Pyro field scale.

4: Extrapolate FLIP Liquid particle geometry's velocity onto temporary field.

5: Add temporary velocity field value to FLIP Pyro's velocity field, multiplying temporary field by an arbitrary fraction to soften influence.

7.2.3 Negate FLIP Pyro Fuel and Temperature

The main two attributes that cause the combustion effect in our FLIP Pyro solver are fuel and temperature. By negating them with the FLIP liquids values, we also end the combustion process and reduce the systems fluid quantities to null with time. The steps to do this are:

For each time-step:

1: Import FLIP Liquid particle geometry, assign local unique identification.

2: Create temporary temperature scalar field.

3: Resize temporary field to match FLIP Pyro field scale.

4: Extrapolate FLIP Liquid particle geometry's temperature onto temporary field.

5: Using a custom Gas Field VOP script, invert the temporary fields values.

6: Multiply an arbitrary fraction of our temporary fields value with the FLIP Pyro's temperature and fuel fields.

7: Feed the values back into the FLIP Pyro's network, just before we create our old fields (solver input 3).

7.2.4 Collision

Collisions are done in a straight forward manner. At every time-step we create a mesh out of our FLIP Liquid particle geometry using the Particle Fluid Surface SOP. We can then import the generated mesh into our DOP network, and assign it as a collision object using the standard method 5.6.

7.3 Fire Spread Across Multiple Solvers

In nature, fire tends to spread from one object to another. Different object may have different chemical compositions, and as such burn in a different manner with varying levels of ferocity. We demonstrates a custom setup that allows us to replicate the effect of fire spreading through an environment across multiple solvers.

7.3.1 Overview

Our system simulates fire spreading from a bunny, to a table, then finally onto a lamp. The way in which the system works is that it records temperature impacts from influencing systems at every time step. Once a certain threshold is passed for a certain point on the affected object, particles are seeded from that point. Whenever an object is activated, i.e. begins seeding particles after passing an arbitrary temperature threshold, it acts as an influence on both the environment objects, and the object from which it was seeded, allowing fire to creep along the objects self without the need for an separate temperature source.

Figure 7.5: Our fire spreading from the bunny to the table and the lamp.

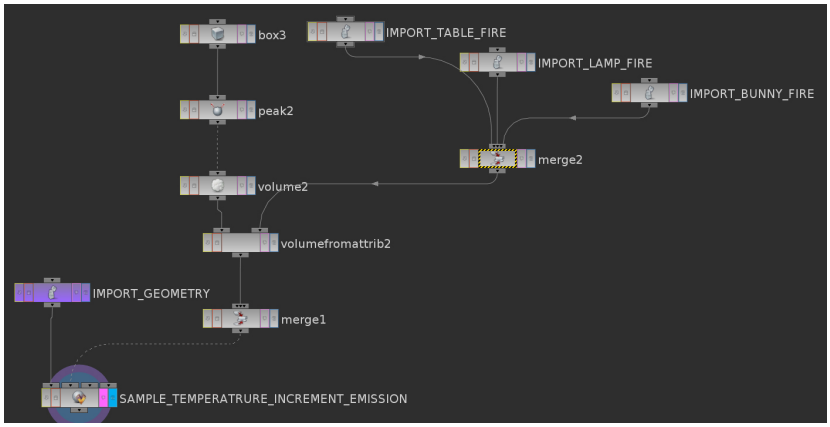

Figure 7.6: We import the FLIP Pyro particles, create a temperature scalar field which we then sample to increment our emission attribute.

7.3.2 Temperature Impacts

Using a custom SOP Solver, we can sample temperature impacts at any geometric point in our scene. The steps necessary in order to achieve this are:

For each time step:

1: Import particle Geometry from all affector FLIP Pyro systems into an external SOP Solver.

2: Copy temperature attributes from particles onto a temporary SOP scalar volume.

3: Using a custom VOP SOP script sample the volumes attributes.

4: (In VOP SOP) If the temperature values exceed a certain value, increment a custom emission attribute on the closest point of our emission geometry.

7.3.3 Particle Seeding

Now at every frame, objects that come into contact with hot flames from one of our scenes FLIP Pyro solvers will have an emission attribute incremented by an arbitrary value at every frame. But that doesn't control anything just yet.

The Particle Fluid Emitter DOP in each of our solvers has the capability of emitting particles from faces depending on their value. However, there are issues with this method, as if we were to feed it two faces, one with an emission value of 0.1, and

the other with an emission value of 1.0, both give very similar results. This in turn causes the whole geometry object to be engulfed in flames immediately, and beyond our control. A more simple and controllable approach to this would be: For each time step:

1: Accumulate emission attributes on affected objects, each step indicating a single impact.

2: Group all faces that have accumulated an arbitrary number of impacts, corresponding to the number of frames it takes to set allight an object under constant impact.

3: Assign any faces in our group with a custom seed attribute with a value of one. Remaining faces are assigned a seed attribute with a value of zero.

4: In our FLIP Pyro particle emitter, set emission to be based on the seed attribute.

The user may now control the time it takes for objects to be set alight, and the time it takes for the fire to spread.

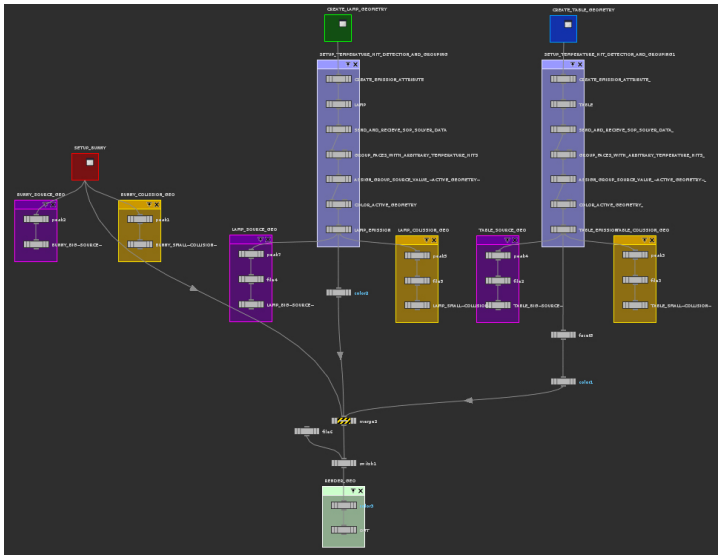

Figure 7.7: Our SOP network, where primitives that have accumulated an arbitrary number of temperature hits is then activated for particle emission in our FLIP Pyro Solver.

8

Conclusion

8.1 Results Analysis

The initial motivation for a FLIP Pyro solver were primarily concerned with : Speed, Resolution, Slow Motion, Memory and Control. The results where not conclusive. However, there is evidence to show that the FLIP Pyro solver can produce a realistic and aesthetically pleasing result at a reduced cost.

8.1.1 Speed

Early on during development, it was noted that a like for like comparison with the Volume Pyro solver was not possible. In the Volume solver, resolution is determined by the grid size, i.e. the number of present cells. In a FLIP Pyro system, resolution is determined by the grid size, the number of particles, the particle scale and the extrapolation distance.

If we increase the number of seeded particles, the simulation times significantly increase, but the resolution increases proportionally. The particle scale and extrapolation distances both affect the amount of influence particles have on one another, and as such the degree of independence present in the particles. Increasing the particle scale and extrapolation distances lowers the resolution, increases viscosity, and vice-versa is true.

Our FLIP Pyro solver resizes dynamically at ever frame, as mentioned in **??**. For large simulations, the Volume solver must start off with a large grid, which severely affects simulations time.

As there are more degrees of freedom in our particles than grid cells, we can output fairly detailed fluid motion on course grids, where the volume solver would output poor details. It is perhaps more appropriate to compare our solver with a much higher resolution Volume solver, which would conclusively point to a significant speed advantage.

That said, the results were not as overwhelming as expected. Particle operators (POPs) are amongst the oldest operators in Houdini, and are generally very slow with many nodes still lacking multi-threading capability. We have two POP solvers in our system, one that seeds particles, and another that manipulates them. During the implementation of the 'Fire Spread' setup, it was observed that every time a large number of particles were impulse seeded into the system, that frame would take approximately 3 to 6 time longer to simulate.

Another probable cause for the underwhelming performance is the process where we copy particles to and from the grid. In the FLIP Liquid solver, this operation is performed two times at every time-step, however in our system this operation is performed ten times at every time-step. The process is quite slow in Houdini, particularly when working with fairly dense grids or a large number of particles.

Finally, and in the absence of a render-time point cloud volumizer, we still need to convert our points into volumes in the SOP context prior to rendering it. This process is very slow, and while it does not affect simulation times, it increases render-times significantly. Furthermore, this process caps the amount of detail we can render from our points.

8.1.2 Resolution

During the implementation of the 'Fire + Water' example, we simulated it on a grid with a division scale of 0.1 units. The particles were then fed into our volumizer, where we were able to extract new details from the system up until a division scale of 0.01 units, at which point we hit a memory limit. Effectively, our particles simulated with 10 fold the grids level of detail.

8.1.3 Slow Motion

Slow motion worked as intended. The only observation noted was that the system may produce slightly varying results as we change the frame rate value, similar to when we increase the number of sub-steps for any of the standard solvers.

8.2 Memory

We have found a very significant memory decrease with the FLIP Pyro solver. This could in retrospect be improved further by deleting unnecessary attributes from the particles prior to storage. Figure4.2 is an illustration of just how significant the memory decrease was found to be.

8.3 Control

We were able to incorporate a POP Solver into our system, which allowed us to apply particle forces and manipulators to our particles at every time-step just prior to solving. Effectively, we can use any of Houdini's particle operations to control the motion and trajectory of our particles, in addition to standard DOP forces and manipulators.

8.4 Future Work

8.4.1 SOP Solver Seeding

One of the ways we can increase the speed of the system is by using SOP Solvers to seed particles. SOP Solvers are fully multi-threaded, and generally much faster than POP Solvers. Houdini's upcoming twelfth release will also see the introduction of a new geometry engine which promises very significant speed increases, a further motivation to abandon POP Solvers in favour of SOP Solvers (SESI, 2011).

8.4.2 Render-time Volumizer

A render-time volumizer is necessary in order for our solver to truly rival the current volume render solution. By using a point cloud lookup in a volume shader at render-time, we should be able to circumvent the slow process of volumizing our particles prior to rendering, as well are removing the current resolution limit.

8.4.3 GPU Simulation

Houdini's upcoming twelfth release allows users to simulate Pyro Volume effects on the GPU (SESI, 2011), a feature present in many of the propitiatory systems discussed in 2.3. By using the new GPU based solvers, we should be able to convert our FLIP Pyro

solver into a GPU based one, allowing us to significantly increase performance, and possibly resolution.

8.4.4 Particle Turbulence

An integral component of pyro simulations is turbulence operations that introduces swirling motions into our fluids (Bridson et al., 2007). This is usually computed on the grid, however, with the grid method we are effectively limited by the grids resolution. If we instead compute turbulence on a particle level we may achieve significantly more detailed motion at distances significantly smaller than the grid cell. This applies to any other force that evolves through space.

8.5 Conclusion

We successfully demonstrated the first FLIP Gas Solver in Houdini, and that it is even possible. Through a set of custom setups, we showed we were able to produce realistic and detailed results. We demonstrated the ability to integrate our solver with a variety of other solvers, proving its extensibility. We were able to control our simulations using particle operations in addition to the standard dynamic operations. We showed that our solver requires minimal secondary storage, and that we can achieve very slow motion simulations with it.

This implementation still requires a number of iterations before our solver is a complete and viable alternative to the current existing off-the-shelf solutions. Much work is needed in order to produce results as seen in current propitiatory systems, and it is the authors full intention to continue development until such is possible.

References

2012 (2009), Film. Directed by Roland Emmerich. USA: Columbia Pictures.

Avatar (2009), Film. Directed by James Cameron. USA: Twentieth Century Fox Film Corporation.

Baratuci, D. (2006), 'What is pressure ?.', Learn Thermo. Available From : `http://www.learnthermo.com/T1-tutorial/ch01/lesson-E/pg02.php` [Accessed 15 August 2011].

Brackbill, J. U. and Ruppel, H. M. (1986), 'Flip: A method for adaptively zoned, particle-in-cell calculuations of fluid flows in two dimensions.', *J. Comp. Phys.* **65**, 314–343.

Bridson, R. (2008), *Fluid Simulation for Computer Graphics.*, A.K Peters.

Bridson, R., Fedkiw, R. and Muller-Fisher, M. (2006), 'Fluid simulation', *SIGGRAPH 2006 course notes* . Available From : `http://www.cs.ubc.ca/rbridson/fluidsimulation/` [Accessed 08 August 2011].

Bridson, R., Houriham, J. and Nordenstam, M. (2007), 'Curl-noise for procedural fluid flow.', *ACM Trans. Graph.* **26**(3).

Clark, J. (2010), 'Houdini 11 review.', 3D World. Available From : `http://www.3dworldmag.com/2010/11/09/houdini-11-review/` [Accessed 08 August 2011].

Desowitz, B. (2009), '2012: The end of the world as we know it.', Animation World Network. Available From : `http://www.awn.com/articles/article/2012-end-world-we-know-it/page/3,1` [Accessed 08 August 2011].

Desowitz, B. (2010), 'Introducing plume for firebending.', Animation World Network. Available From : `http://www.awn.com/articles/visual-effects/introducing-plume-firebending` [Accessed 08 August 2011].

F.A.A., F. A. A. (1975), *Aviation Weather.*, Aviation Supplies Academics, Inc.

Failes, I. (2010), 'Ilms elements for the last airbender.', fxguide. Available From : `http://www.fxguide.com/featured/ILMs_Elements_for_The_Last_Airbender/` [Accessed 08 August 2011].

Ghourab, A. (2011), 'Ahmad ghourab showreel 2011.', Vimeo.

Harlow, F. H. (1963), 'The particle-in-cell method for numerical solution of problems in fluid dynamics.', *Experimental Arithmetic, High Speed Computations and Mathematics.* pp. 269–269. Providence, RI: American Math. Society.

Harlow, F. H. (2004), 'Fluid dynamics in group t-3 los alamos national laboratory.', *J. Comput. Phys.* **195**(2), 414–433.

Harris, M. (2005), *Fast Fluid Dynamics Simulation on the GPU.*, Addison-Wesley Professional.

Hovrath, C. and Gieger, W. (2009), 'Directable, high-resolution simulation of fire on the gpu.', *ACM Trans. Graph.* **28**(3).

Klosters, C. (2009), 'Masterclass: Pyro fx.', Side Effects Software. Available From : `http://www.sidefx.com/index.php?option=com_content&task=view&id=1496&Itemid=166` [Accessed 08 August 2011].

Lait, J. (2010), *Flip Fluid Masterclass.*, Side Effects Software. Available From : `http://www.sidefx.com/index.php?option=com_content&task=view&id=1834&Itemid=166` [Accessed 08 August 2011].

Nordenstam, M. (2010), 'Naiad 0.4.2 release notes.', Exotic Matter Support. Available From : `http://exoticmatter.zendesk.com/entries/310914-naiad-0-4-2-release-notes/` [Accessed 15 August 2011].

Nordenstam, M. (2011), 'News.', Exotic Matter. Available From : `http://www.exoticmatter.com/` [Accessed 15 August 2011].

Nordenstam, M. and Bridson, R. (2008), 'Naiad [computer program]', Exotic Matter. Available From : `http://www.exoticmatter.com/company/` [Accessed 08 August 2011].

Priscott, C. (2010), 3d langrangian fluid solver using sph approximations., Master's thesis, N.C.C.A. Bournemouth Univesity.

Rutter, G. (2011), 'Liquids.', Chemistry Explained. Available From : `http://www.chemistryexplained.com/Kr-Ma/Liquids.html` [Accessed 15 August 2011].

Self, B. (2009), 'Houdini 10 blasts off with pyro fx tools progressive ipr and stereoscopic 3d.', Side Effects Software. Available From : `http://www.sidefx.com/index.php?option=com_content&task=view&id=1478&Itemid=55` [Accessed 08 August 2011].

Self, B. (2010), 'Side effects software releases houdini 11 faster, easier and more productive.', Side Effects Software. Available From : `http://www.sidefx.com/index.php?option=com_content&task=view&id=1772&Itemid=55` [Accessed 08 August 2011].

SESI (2011), 'Sneak peek houdini 12.', Side Effects Software. Available From : `http://www.sidefx.com/index.php?option=com_content&task=view&id=2005&Itemid=66` [Accessed 08 August 2011].

Seymour, M. (2010), 'The tech behind the tools of avatar part 2: Naiad.', fxguide. Available From : `http://www.fxguide.com/featured/The_Tech_Behind_the_Tools_of_Avatar_Part_2_Naiad/` [Accessed 08 August 2011].

Stam, J. (1999), 'Stable fluids.', *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 99).* **24**(3), 121–128.

The Last Airbender (2010), Film. Directed by M. Night Shyamalan. USA: Paramount Pictures.

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

9

User Guide

Please refer to the scene files submitted with this report for the basic set-up of the DOP Network.

Option	Functionality
Smoke Amount	Emit smoke from source object
Fuel Amount	Emit Fuel from source object
Scale Temperature	Temperature scale from source object
Add Noise	Adds noise to source object. Creates non-uniform sourcing.
Force Scale	Scales particle attributes update amount
Viscosity	Changes the ratio of FLIP to PIC. 0 is purely FLIP, 1 Is purely PIC.
Cooling Rate	The rate at which temperature is decremented at every frame
Buoyancy	The total buoyancy force
Buoyancy Direction	Buoyancy direction control
Vortex Confinement	Larger values increases curl forces in the system
Separation	Controls particle separation scale and operations
Ignition Temperature	The temperature required before combustion takes place
Burn Rate	The amount of fuel used for every ignition
Soot Smoke Rate	Amount of density released for every unit of fuel consumed
Temperature Output	The amount of temperature released for every unit of fuel consumed
Gas Released	The amount of gas released, outward force for every unit of fuel consumed
Heat Cool Time	The number of seconds it takes heat to cool from 1 to 0.
Diffusion	The blur scale applied to each of the scalar fields at every frame
Turbulence	Introduces curl noise into the system
Volume Limit	The maximum size and position the simulation may occupy
Extrapolation	The cell distance at which particle data is extrapolated to
Vorticle Strength	Overall scale of vorticle forces.
Clear	User may choose which fields to clear at every frame to save memory