

**The Implementation of 2D Fluid Solver plug-in for
Houdini8.0**

Mya Yee Win

A thesis submitted for the degree of
M.Sc Computer Animation
September 11, 2007

National Centre for Computer Animation
Bournemouth University

List of Figures

A simulation grid to store the density and velocity values	9
2D illustration of how the velocity is stored in a staggered grid.....	9
Illustration of the cells reserved for controlling boundary values.....	10
Simulation process of fluid solver.....	12
Basic structure of the density solver.....	14
A parameter window of the plug-in.....	16
A fluid 2D solver SOP node.....	16
Illustration of the data flow between processes.....	17
Turbulent wave (A screenshot from houdini).....	18
Smoke (A screenshot from houdini).....	18

Chapter 1. Abstract

There are numerous algorithms and techniques for fluid simulation. Among them, some of the algorithms are simple, easy to implement and also produce good results. The purpose for this thesis is about developing a 2D fluid solver plug-in using existing algorithms for houdini users. The discussion about the pros and cons of the plug-in is also included at the end of the thesis.

Chapter 2. Introduction

Most of the people these days are quite familiar with cinematic special effects like fluid effects, in the form of smoke, water, fire, or wind. One of the possible ways would be to film fluid effects live on set. But in reality, it is not that simple because many movie makers demanded fluid effects that are far beyond than reality. Sometimes these effects cannot be feasible, affordable, and safe to film on set. Animating the physical phenomena such as water, fire and smoke is quite complicated to create using the traditional methods. That is why simulating fluids has become popular in computer graphics but it still remains a challenging problem. This is not surprising since the motion of gases such as smoke is highly complex and turbulent. Ideally, a good CG fluid or smoke model should both be easy to use and produce highly realistic results. Over the decades, people have been researching and developing new technologies for the fluid simulation. However, it's been found out that not many people have been using those available resources but instead they are trying to reinvent which have been discovered. The purpose for this thesis is to create a fluid solver plug-in using those existing resources, and making them available for the artists and the technical directors.

This thesis is organized in the following order:

Chapter 1 presents the abstract of the thesis.

Chapter 2 presents the introduction of the thesis.

Chapter 3 introduces brief information about the commercial fluid software packages and how different CG houses have been using them. Following that information are different algorithms for the fluid simulation.

Chapter 4 presents the implementation details about the fluid solver and the plug-in and shows the result.

Chapter 5 presents a discussion about the plug-in and the conclusion of the thesis.

Chapter 6 presents the future work.

Chapter 7 presents the appendices where main functions will be explained in details.

Chapter 8 concludes with the bibliography.

All the source codes, HTML files of the source codes generated by doxygen, plug-in, pdf documentation of this thesis, user guide in HTML format, and some rendered images and sequence are presented in the CD with the thesis.

Chapter 3. Previous Work

3.1 Introduction

The big CG houses have some amount of proprietary CFD software, for example ILM has their own proprietary software[25] but the smaller houses usually rely on plug-ins and outsourcing. Since cost and visual impact are higher priority in CG, non-CFD solutions are typically preferred. In most cases, visual effects producers require a simple solution that runs on a desktop PC. There are a few commercial packages for the fluid; for instance, Realflow and Flowline.

RealFlow is a fluid simulation software that utilizes particle-based fluid dynamics and implicit surfaces for rendering realistic fluid dynamics. Flowline is an in-house developed tool created by Scanline for creating realistic fluid simulations. The Moving Picture Company has recently licensed Flowline and integrated into its production pipeline[26].

3.2 Fluid Simulation Algorithms

The first physically based simulation of complex water effects using the full 3D Navier-Stokes equations has been based upon the large amount of research done by the computational fluid dynamics community (CFD) over the past 50 years. Foster and Metaxas[12] utilized the work of Harlow and Welch [3] in developing a 2D and 3D Navier-Stokes methodology for the realistic animation of liquids. A semi-Lagrangian “stable fluids” method was introduced to the computer graphics community by Stam[6] in order to allow the use of significantly larger time steps without losing stability. Foster and Fedkiw[14] made important contributions to the simulation and control of three dimensional fluid simulations through the introduction of a hybrid liquid volume model combining implicit surfaces(Level Set) and massless marker particles.

In order to simplify the simulation, the fluid is assumed to be homogeneous and incompressible. The homogeneity part implies that the simulation only treats one kind of fluid, and not multiple fluids and their interactions. Incompressibility means that the

density of the fluid is constant. Given these assumptions, the Navier-Stokes equations for homogeneous incompressible fluids [6] can be used.

The equations are defined as follows:

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

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

\mathbf{u}	=Velocity field
t	=Time
p	=Pressure field
ν	=Viscosity
\mathbf{f}	=Force field

3.2.1 Solving the Equations

These two equations state that the velocity should conserve both mass (Equation 1) and momentum (Equation 2)[6]. The first equation, known as the continuity equation, states that the divergence of the velocity must always be equal to zero. This constraint ensures that the sum of all velocities entering and leaving at a point in space is zero. When this is conserved, the total mass is conserved over the entire simulation.

The second equation, known as momentum equation, describes the change of the velocity in a fluid. The change of velocity involves 3 computations to update the velocity at each time step:

- (i) advection,
 - (ii) diffusion and
 - (iii) external forces application
 - (iv) solving the forces formed by the pressure.
- (i) Advection

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

The first term of the equation(2), advection, represents the fact that the motion of the fluid causes motion of the entities contained within it. This can be thought of as the velocity of the fluid moving itself along, and is sometimes referred to as self-advection because of that property. The advection part of the equations can also be used to model motion of other entities inside the fluid.

(ii) Diffusion

$$\nu \nabla^2 \mathbf{u}$$

Different kinds of fluids behave differently. Water for instance, reacts very rapidly and lively, when a force is acting on it. Paint or syrup on the other hand is much more resistant to interaction when a force is acting on these liquids. This fluid property is called viscosity, ν , and is modeled by the diffusion part of the equation. This part of the equation allows the velocity to propagate outwards from its current location, with the viscosity parameter controlling how fast this happens. High viscosity yields thick and slow fluids, while a low yields lively fluids. As smoke is lively and active, it is similar to water. As the viscosity goes toward zero, the contribution from it will be smaller and smaller, and therefore be less likely to give any contribution to the overall simulation. Therefore, for this project, the viscosity part would be to leave out from the diffusion part completely.

(iii) Forces (f)

The third term of the equation(2) is considered as the sum of all external forces. Natural forces, like the wind blowing smoke and the drag generated by gravitational forces are external forces. Some of these forces are easily simulated by constant forces, such as the gravity, while others require separate simulations, such as the forces generated by temperature differences. Another source of external forces could be the user interacting with the simulation, for instance controlling a fan or a hand inside a simulation.

(iv) Pressure

$$\nabla p$$

The last term, pressure, presents in the equation(2), have an influence on the motion of the fluid. This can be considered as the fluid from the area with high pressure will be pushed by the pressure toward the area with lower pressure. This force is represented by the gradient of the pressure field, p .

3.2.2 Solving the Equations Numerically

Turning away from the infinite equations described by the physical relations, a discrete representation must be selected. The numerical methods used when solving the equations are closely tied to the discrete representation, as it attempts to approximate the equations using the data present. The discrete representation is used for storing the different fields of the simulation, velocity, pressure, density, temperature, color and external forces. Currently, numerical methods are generally grouped in to two categories, Lagrangian method (particle-based) and Eulerian method (grid-based).

Lagrangian Method

Lagrangian methods represent the fields by small massless placeholder particles, which are moved around during the simulation. Each particle is assigned a number of quantities, such as velocity, temperature, etc. The particles are then used to approximate the necessary values at arbitrary locations in the field, by using particles close to the desired location. Similarly, when calculating derivatives, the lagrangian methods use particles within some range to generate approximations.

Eulerian Method

Eulerian methods take a quite different approach. The second approach, the grid-system, divides the space a number of rectangular cells, with each cell storing the relevant quantities inside it, velocity, density, pressure, etc. The desired quantities can be read directly from the cells, and derivatives can be approximated by using values stored in neighboring cells.

The two different approaches have different advantages and disadvantages, and both are useful in different cases.

Advantages and Disadvantages of Lagrangian and Eulerian Approaches

The representation of particles in lagrangian methods is good when simulating more than one fluid. For example, when simulating water, it is necessary to determine the surface that separates water and air. The particles are useful when determining the surface. This approach is generally considered not a practical method to do for computer graphics, due to the large amount of particles needed to be rendered on the screen in order for it to be realistic; however this approach is often used by the physics people due to it being more physically real.

Eulerian methods have the advantage of having an efficient representation. The performance is independent of the amount of matter introduced in the simulation. However, as the space simulated is to be divided into cells, the growth of the time complexity becomes cubic in three dimensions, and thus limits the size of the volume which can be simulated.

Among those two methods, Euler's equation is considered to be much faster. Therefore Euler's equation for the fluid simulation is used for this project.

3.2.3 Representing the Cells

When all quantities are stored at the center of each grid cell, the grid is known as a collocated grid. There is another type of grid called a staggered grid which stores the velocity on the center of the faces of each grid cell instead, while still keeping the other properties at the center. The partitioning of space is restricted to cells with the same size across the entire simulated area. The simulation grid is split into cells along each of the dimensions in width \times height, with a length of h . The grid is indexed like using integer coordinates going from 0 to $N + 1$.

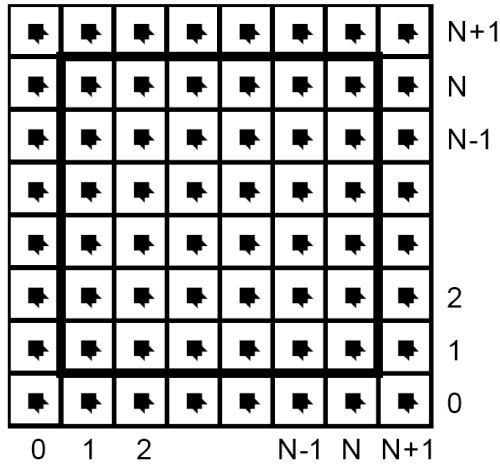


Fig1. A simulation grid to store the density and velocity values. These values are stored at the cell centers. The grid contains an extra layer of cells to account for the boundary conditions. *Image taken from Jos Stam's Real-Time Fluid Dynamics for Games*

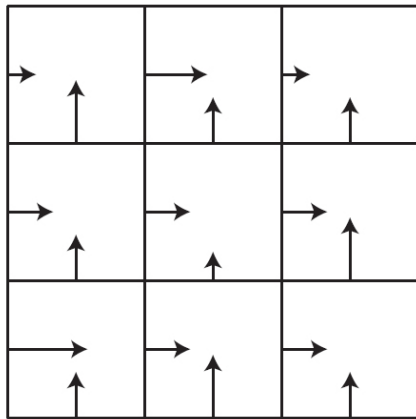


Fig2. 2D illustration of how the velocity is stored in a staggered grid.

3.2.4 Finite Differences

In order to evaluate the equations, a method to approximate the derivatives from values in a discrete grid is needed. Taylor's theorem can be used to state the derivative of an arbitrary function, f , at parameter value x_0 with arbitrary precision[6].

Assume having a function of two variables: $f(x, y)$

Finite difference approximations to mixed partial derivatives can be found by using Taylor's theorem: Assuming that Dx is the grid spacing (always positive) that can be

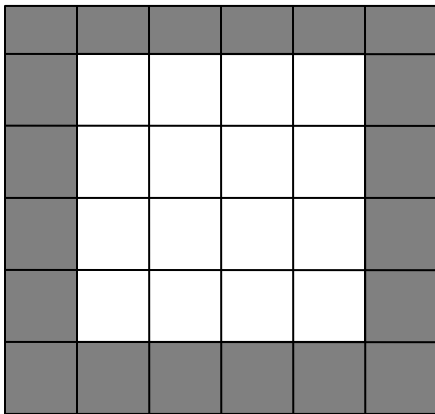
formed finite differences like this: $\frac{\partial^2 f}{\partial y \partial x}(a, b)$

$$\frac{\partial f}{\partial x}(a, b) \approx \frac{f(a+\Delta x, b) - f(a-\Delta x, b)}{2\Delta x}$$

$$\frac{\partial f}{\partial y} \left(\frac{\partial f}{\partial x} \right) (a, b) = \frac{\frac{\partial f}{\partial x}(a, b+\Delta y) - \frac{\partial f}{\partial x}(a, b-\Delta y)}{2\Delta y} = \frac{f(a+\Delta x, b+\Delta y) - f(a-\Delta x, b+\Delta y) - f(a+\Delta x, b-\Delta y) + f(a-\Delta x, b-\Delta y)}{4\Delta x \Delta y}$$

3.2.5 Boundary Conditions

A boundary conditions an essential topic regarding the simulation. A boundary is defined to be the face between two simulation cells, where one of the cells is participating in the simulation, and the other is not. These boundaries appear at the edge of the simulated area, and it needs to be determined how the various fields behave, in order to control the simulation. This is necessary, as it can have a large impact on how the simulation behaves close to the boundary. To control the values of the fields near the border of the simulation, all the cells of the simulated area having one ore more faces that borders the outside of the simulated area, are reserved as boundary cells. The boundary cells are cells which are not used for simulation, but instead are used for controlling the values between the boundary cells and their neighboring non-boundary cells.




 Border boundary cell

Fig3. Illustration of the cells reserved for controlling boundary values

Chapter 4. Development of 2D fluid solver plug-in

4.1 Introduction

There are a lot of resources out there for the fluid simulation and nobody has made a good use of those resources yet. Therefore the author has come up with an idea how to integrate the existing available fluid solvers into the existing 3D software efficiently? Among these available resources, the programs and algorithms distributed by Jos Stam are available for the public and easy to implement. *“Jos Stam is a leading researcher in the field of computer graphics, focusing on subdivision surfaces, rendering algorithms and the simulation of natural physical phenomena, Stam made significant contributions to the fluids simulation component of Alias' Maya 3D content creation software product. Stam is now a "Senior Research Scientist" at Autodesk, Inc.”*[24]

The third party softwares like RealFlow and Flowline are really powerful and can give convincing look but the problem with them is for the CG houses, they have their own production pipeline and there are no proven pipelines to run and render in the way it is expected to work. Therefore, to fit into the production pipeline could be challenging, time consuming and expensive. After a few research done about 3D softwares, it appears that Houdini would be a good choice for this project. Houdini provides a good node-based work flow system and also Houdini Development Kit is a C++ based which can be used to develop any types of nodes. The current version (which is 8.0) of Houdini does not have the fluid solver yet but with fluid solver coming in Houdini 9.0, it would be very interesting. The official release date for Houdini9.0 is unknown yet and usually for the CG industry, the transition period from the old version to the new software could be quite long. Meantime, to produce fluid like motion, writing a custom plug-in seems to be an ideal solution. Because of the node- based structure of the Houdini and lack of fluid solver for the current version, Houdini is chosen and fluid solver plug-in is written in HDK. Based on the algorithms written by Stam, a fluid solver in C++ has been developed and written a plug-in using hdk for making an interface in houdini. Meantime, if anyone comes across to do their FX works using fluid simulation, this plugin will help them to solve the problem.

After the software has been decided, a new area of research needs to be done again to make the plug-in efficient. The design of plug-in has been carefully planned and made with the help of the TD's.

4.2 Implementation Details

Stages of developing the custom fluid solver plug-in in Houdini

1. Implementation of fluid solver in C++
2. Implementation of GUI for the plug-in in HDK
3. Integrate the solver in HDK and create a custom node

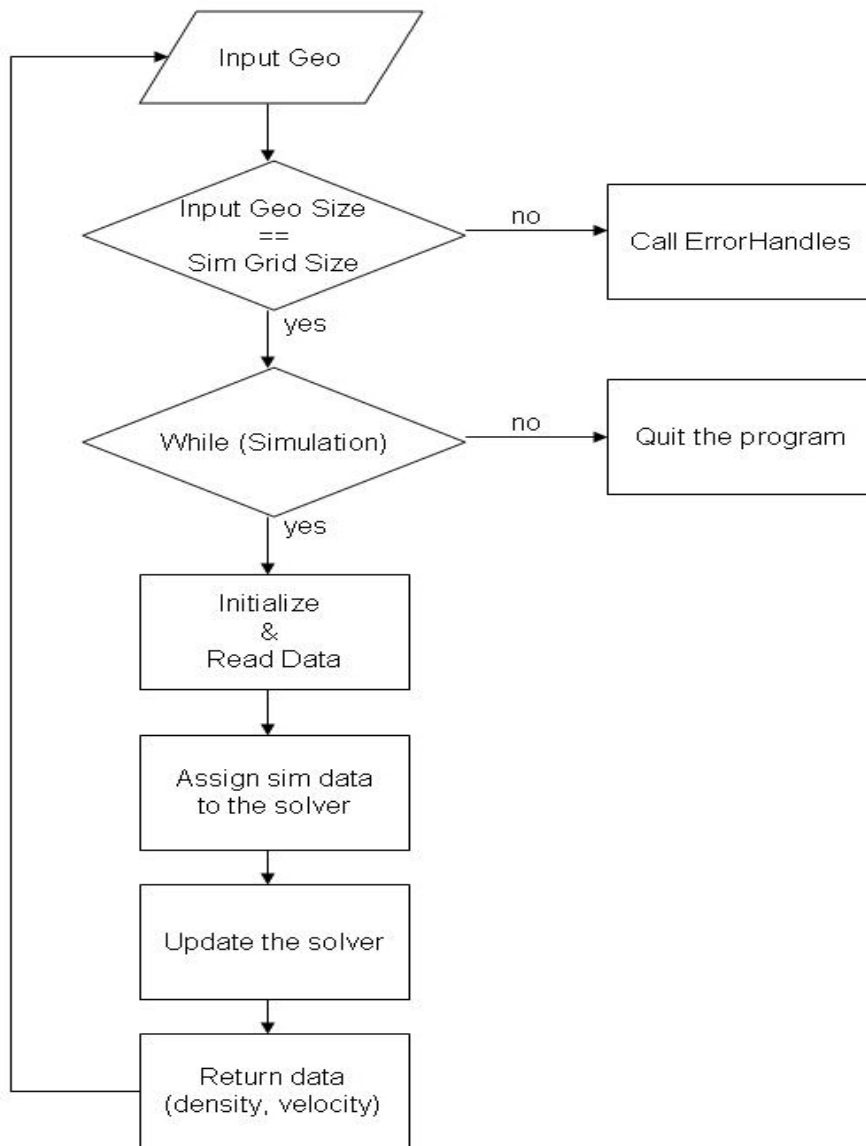


Fig4. Simulation process of fluid solver

4.2.1 Implementation of the fluid solver in C++

The solver is based on the Jos Stam's Realtime Fluid 2D solver and is written in C++. Extra features like adding the vorticity confinement force back into the solver. A well known problem with fluid dynamics solution is that they tend to dampen any swirling motions in the velocity field. In other words, the curl of u diminishes too quickly as the simulation progresses. To counteract the undesired dampening of swirling motions, Fedkiw[17] adopted a technique known as vorticity confinement from the CFD literature. The idea is to isolate and increase swirling motions in the fluid by amplifying the curl of the velocity field. During a simulation, the vorticity confinement force is added.

All the technical details have been explained in the technical background and below are the basic pseudocodes for the simulation is like this:

WHILE (simulation)

 Get forces and sources from the user

 Update the density

 Update the velocity

 Output the data back to the scene

ENDWHILE

Get Forces and Sources from the user

The values for the forces and the amount of the density will be input by the user via Houdini program.

Solving the density part of the equation

For moving smoke density, from equation(4), there are four things to consider:

- 1.Add Density Sources
- 2.Advect Density
- 3.Diffuse Density
- 4.Dissipate Density

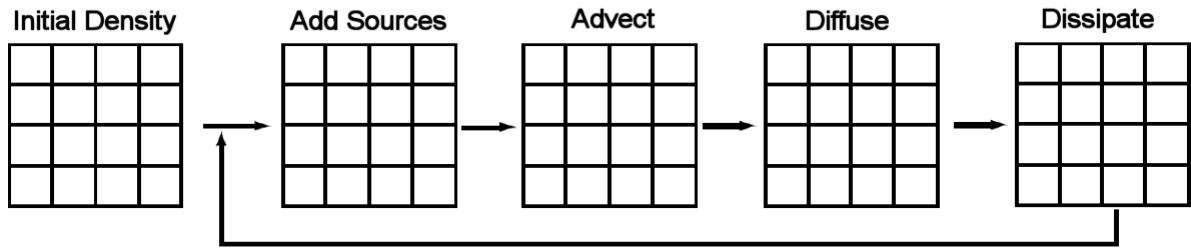


Fig5. Basic structure of the density solver. The three terms appearing on the right hand side of the density equation are solved at every time step. *Image taken from Jos Stam's Real-Time Fluid Dynamics for Games*

Below is the pseudocode for calculating density:

```

PROCEDURE CALC_DENSITY
FOR (number of simulation iterations)
addDensitySource();
advectDensity();
diffuseDensity();
dissipateDensity();
ENDFOR
END

```

Solving the velocity part of the equation

The change in velocity from equation (3) is due to the three terms: advection, diffusion, and forces, right hand side of the equation. There are also a few issues to consider for the velocity to be mass conserving. This is an important property of real fluids because it forces the fluid to have swirly effects. Numerical dissipation is also another important thing to consider for the fluid simulation. Vorticity confinement method, proposed by Fedkiw [6], re-injects the lost energy due to numerical dissipation back into the fluid, through a force which encourages the flow to display small scale vorticity.

```

PROCEDURE CALC_VELOCITY
FOR (number of simulation iterations)

```



```
addDensitySource();
calcVorticityConfinementForce();
diffuseDensity();
ProjectToMakeVelocityConserveMass();
advectDensity();
ProjectToMakeVelocityConserveMass ();
ENDFOR
END
```

4.2.2 Implementation of GUI of the plug-in in HDK

In Houdini, nodes are organized into nested hierarchies which make it easy to step back and make changes, revise, rewire and share among the colleagues. Houdini technical directors are able to build up complex systems and even create custom tools by working interactively in Houdini's network editor without writing any code. C++ based Houdini Developer's Kit is freely distributed with Houdini Apprentice version which allows the developers to extend the software. With HDK, the freedom of adding new tools is infinite. All types of new nodes can be created: SOPs (Surface Operators), COPs (Composition Operators), Objects, CHOPs (Channel Operators), ROPs (Render Operators), DOPs (Dynamic Operators) and POPs (Particle Operators). There are several node types based on their functions like surface operators (SOPs), particle operators (POPs), shader operators (SHOPs) etc. Each operator has an output that can be passed to the next operator as input. Operators can optionally have one or more inputs depending on the type of the nodes, for example, the create type does not need an input but for the modification type nodes needs at least one or two inputs depending on the nature of the node.

GUI is very important for the custom plug-in because that can put off the artist and technical directors' abilities very easily for them to use. Also some technical terms can easily scare the users away. At the same time since this plug-in is designed for the Houdini users, the naming standard and convention should also be considered and should follow the Houdini naming convention and its program structure because that will save a big time for the first time users to figure out what the plug-in does. Considering all these

facts, simple terms are used and multiple user controllable variables are added for flexibility.

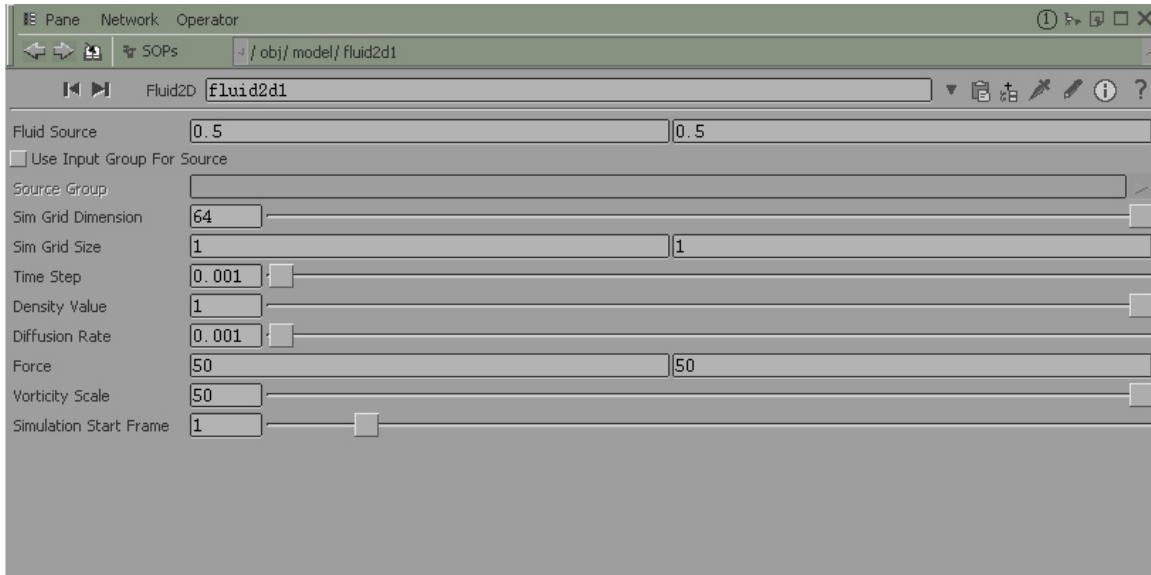


Fig6. A parameter window of the plug-in

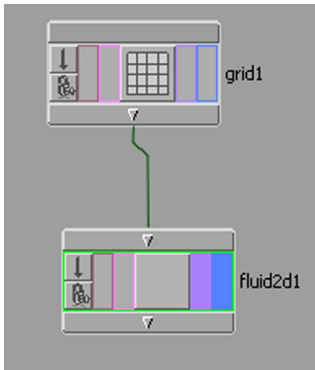


Fig7. A fluid 2D solver SOP node

4.2.3 Integrate the solver in HDK and create a custom node

This section contains details about how developers can widen the power of the software using third party plug-ins like fluid solver. It is implemented in Houdini SOP level and can integrate with any other components.

The simulation parameters are read from the Houdini program. These parameters are passed down to the solver and the solver does all the calculation. After the calculation, the solver gives back density values and velocity values. Those values can be used

anywhere in the creation of different effects, for example, density could be used as the height map for the displacement shader or the color values for the texture map.

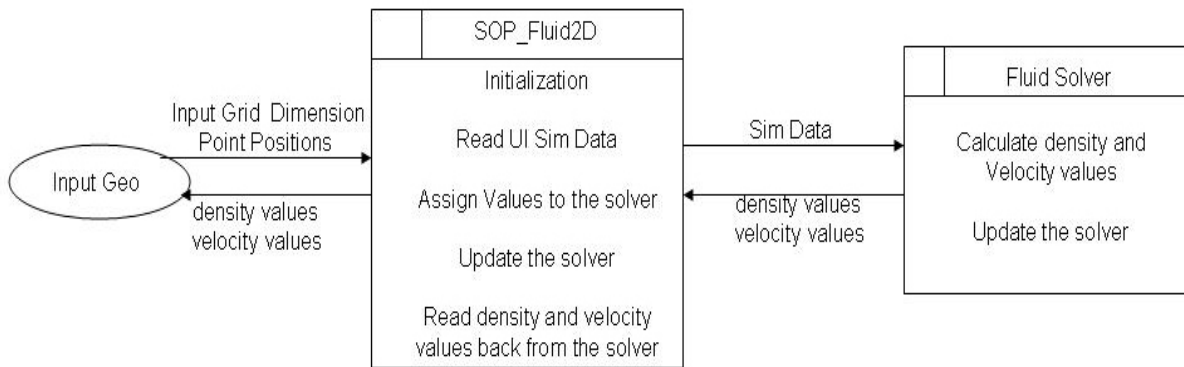


Fig 8. Illustration of the data flow between processes

4.3 Results

This section is about how different effects can be created using the plug-in. The plug-in can be used as a modeling tool and/or animation tool. The artists can define the sources of water and as well as sources for the smoke. Extensive controls allow the artists to modify the simulations and the adjustments of fluid properties at any points in space. With this plug-in the users can now focus more on creating the effects to resemble realism and natural complexity. More exciting effects can be achieved by a few tweaking and adjusting the controls. Here are only a few examples of how this plug-in can be used.

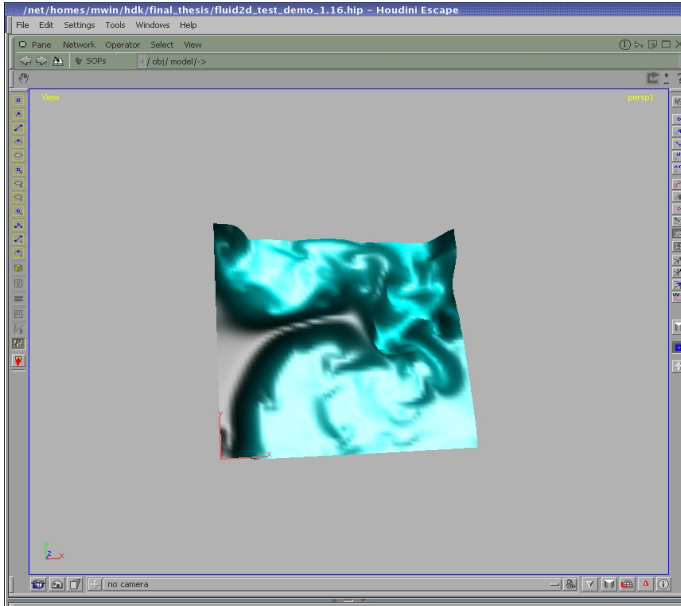


Fig9. Turbulent wave (A screenshot from houdini)

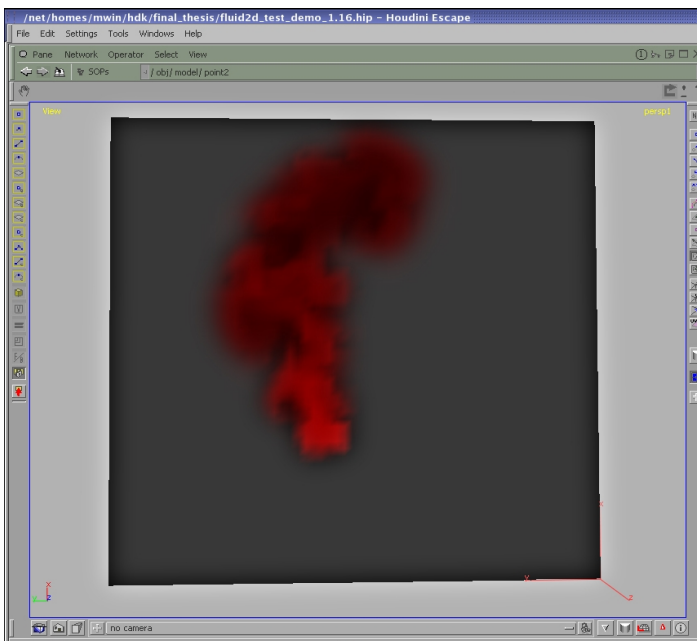


Fig10. Smoke (A screenshot from houdini)

Chapter 5. Discussion and Conclusion

5.1 Discussion

Fluid animation in computer animation is always a challenging problem. Furthermore, to get the complicated look that the directors need and at the same time to keep the natural behaviour and look of water and smoke are even more challenging. For the traditional animators to manually hand animate the fluid animation is not possible. Even if it is possible, it is going to take a large amount of time to get the look that they have desired. This plug-in provides greater artistic control and flexibility. But to be able to get to the desired direction, a few parameters need to be tweaked and fine-tuned until it reaches to the specific look that the artists have intended. The solver is implemented in a very specific way based on the needs of the technical directors and effects artists, which will definitely help their work flow faster. Using this plug-in can also reduce the expensive computational time as well. The simulation time is relatively quick compared to the other fluid solvers since it is based on the requirements of the users and unnecessary calculations are removed. The solver is written in C++ therefore it is highly extensible and any new features can be added or subtracted anytime. With the cooperation of artists' creativities and the flexibility of this plug-in, any interesting effects can be created.

Understanding how fluid is modeled for CG can be a challenge too. This is especially true for students who have not taken courses in vector calculus and differential equations, or are rusty in these subjects. First of all, beginners tend to get slowed down or could even could get completely backed out in the notation of the Navier-Stokes equations and the difficulties that arise when trying to discretize them. Second, the implementation details of the fluid flow systems described in the papers are referenced from a lot of "previous work".

The system has been developed using Houdini8.0 HDK Toolkit therefore it can only be compiled using that version. Although HDK is a very good toolkit, lack of good documentations in it is a major problem. It takes some time to understand how it works. But thankfully Houdini community is very supportive, and some codes are available to look at, it is therefore possible to write a plug-in within a few months.

There are two major types of solving fluid: Eulerian and Lagrangian method. And so far only Eulerian method is efficient in terms of speed and feasibility. Since Eulerian is a grid based approach, it is limited to have the grid input. Its advantage can also be viewed as its disadvantage. For example, once the dimension of the simulation grid is increased, the calculation time went up immediately noticeably and slows down the calculation. Because all the grid points are used (although some of the points are not needed for the simulation) to store all these attributes for example, velocity and density, it takes up a lot of memory space. Sometimes, Houdini cannot handle the large amount of grid numbers and it crushes the program immediately. Last but not the least this plug-in is tested in every possible way and fixed any bugs that came up but the end users might have different experiences and might have found a few bugs that needs to be fixed. Even the fluid simulation software used by ILM which was and is still in use is rewritten over and over again so that it can produce better results within short amount of time. To achieve a reliable and high fidelity fluid solver took many years of work.

5.2 Conclusion

This plug-in is not intended to give the physically accurate fluid simulation. Instead it is meant to work with other surface operator nodes in Houdini and to improve the workflow of the technical directors and effects artists. For most of the time in the visual effects industry, visual quality is more important than the physically accurate animation. With this plug-in, the visual quality produced by using with other operators is also quite convincing. The purpose of this thesis is to use the available resources and make it possible for Houdini users to use the full extent of those resources. Since it has reached its purpose which is to help the technical directors and effects artists to get the fluid like motion within a short amount of time with the extensible user controls, this project can be considered as a success.

Chapter 6. Future work

There are still a few areas that can be improved or needs to be considered:

Extend the solver from 2D to 3D

That can give users more flexibility.

Collision Objects or Obstacles

It needs to be considered how the fluid would react when it collides with stationary or animating objects.

Grid Dimension

Instead of calculating all the points from the simulation grid, calculate only the points within the area where the actual simulation takes place.

Memory Handling

To consider different approaches in memory allocation so that it can prevent Houdini from stopping the program abruptly. This is directly related to the dimension of the simulation grid as mentioned in above.

Boundary Conditions

It would be a good control for the users to have an option like open or close boundary.

Appendix A:

List of main functions from the fluid solver

void diffuse (int b, float *x, float *x0)

-Calculate the diffusion part of the equation. Calculate the input array with diffusion effects. This is achieved through use of a linear solver.

void advect (int b, float *d, float *d0, float *u, float *v)

-Calculate the advection part of the equation. Calculate the input array after advection. Start with an input array from the previous timestep. For all grid cells, it needs to calculate for the next timestep and trace the cell's center position backwards through the velocity field. Then, interpolate from the grid of the previous timestep and assign this value to the current grid cell.

void project (float *u, float *v, float *p, float *div)

-This function is for making the velocity a mass conserving, incompressible field. It forces the flow to have many vortices which produce realistic swirly-like flows. Achieved through a Hodge decomposition: every velocity field is the sum of a mass conserving field and a gradient field. Calculate the divergence field of the velocity first by using the mean finite difference approach, and apply the linear solver to compute the Poisson equation and obtain a "height" field. Then subtract the gradient of this field to obtain the mass conserving velocity field.

void linearSolver (int b, float *x, float *x0, float a, float c)

-Iterative linear system solver using the Gauss-Seidel relaxation technique.

void vorticity_Confinement (float *_u, float *_v)

- Semi-lagrangian approaches suffer from excessive numerical damping. Vorticity confinement method is proposed by Fedkiw [5] to re-inject the lost energy due to numerical dissipation back into the fluid, through a force which encourages the flow to display small scale vorticity. This function calculates the vorticity confinement force for

each cell in the fluid grid. At a point (i,j) , $\text{ForceVorticityConfinement} = \mathbf{N} \times \mathbf{w}$ where \mathbf{w} is the curl at (i,j) and $\mathbf{N} = \nabla |\mathbf{w}| / |\nabla |\mathbf{w}||$. \mathbf{N} is the vector pointing to the vortex center and add force perpendicular to \mathbf{N} .

Appendix B:

List of main functions from the SOP_Fluid2D

void initSystem()

-Initialize the variables and allocated the memory

void readUIData(float currentTime)

-Read data input by the users, these values will be evaluated every frame of the simulation.

void assignValues()

-Assign the read values to the solver.

void Update()

-Call the update functions from the solver; this is where the actual calculation for the solver takes place.

void getDensity()

-Read the density values back from the solver and assign it to each point

void getVelocity()

-Read the velocity values back from the solver and assign it to each point

Bibliography

- [1] D. Enright, S. Marschner and R. Fedkiw, Animation and Rendering of Complex Water Surfaces, in SIGGRAPH 2002 Conference Proceedings, Annual Conference Series, July 2002, 736-744.
- [2] D. Q. Nguyen, R. Fedkiw and H. W. Jensen, Physically Based Modeling and Animation of Fire, in SIGGRAPH 2002 Conference Proceedings, Annual Conference Series, July 2002, 721-728.
- [3] Harlow, F.H., and Welch, J.E., "Numerical Calculation of Time-Dependent Viscous Incompressible Flow," *Phys. Fluids*, 8, 1965, pp. 2182-2189.
- [4] J. Stam and D. Brinsmead, Method of Producing Fluid-Like Animations Using a Rapid and Stable Solver for the Navier-Stokes Equations, U. S. Patent#6,266,071 B1, July 24, 2001.
- [5] J. Stam, A General Animation Framework for Gaseous Phenomena, ERCIM Research Report R047, January 1997.
- [6] J. Stam, Stable Fluids, In SIGGRAPH 99 Conference Proceedings, Annual Conference Series, August 1999, 121-128.
- [7] J. Stam, Interacting with Smoke and Fire in Real-Time. *Communications of the ACM*, Volume 43, Issue 7, 2000, 76-83.
- [8] J. Stam, A Simple Fluid Solver based on the FFT, *Journal of Graphics Tools* Volume 6, Number 2, 2001, 43-52.
- [9] Jos Stam, Real-Time Fluid Dynamics for Games. *Proceedings of the Game Developer Conference*, March 2003.

- [10] Kuldeep Singh, Engineering Mathematics Through Applications, Palgrave Macmillan, 2003
- [11] M. Van Dyke, An Album of Fluid Motion, The Parabolic Press, Stanford, California, 1982.
- [12] N. Foster and D. Metaxas, Realistic Animation of Liquids, Graphical Models and Image Processing, volume 58, number 5, 1996, 471-483.
- [13] N. Foster and D. Metaxas, Modeling the Motion of a Hot, Turbulent Gas, In SIGGRAPH 2001 Conference Proceedings, Annual Conference Series, August 1997, 181-188.
- [14] N.Foster and R. Fedkiw, Practical Animation of Liquids, In SIGGRAPH 2001 Conference Proceedings, Annual Conference Series, August 2001, 23-30. The Iterative Methods Library (IML++), developed by NIST, available at <http://math.nist.gov/iml++/>.
- [15] Randima Fernando, GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics, Addison-Wesley Professional, 2004.
- [16] R. Courant and E. Isaacson and M. Rees, On the Solution of Nonlinear Hyperbolic Differential Equations by Finite Differences, Communication on Pure and Applied Mathematics, 5, 1952, 243-255.
- [17] R. Fedkiw, J. Stam and H. W. Jensen, Visual Simulation of Smoke, In SIGGRAPH 2001 Conference Proceedings, Annual Conference Series, August 2001, 15-22.

[18] Houdini9 Public Beta News Article. Accessed on August, 2007.

<http://www.sidefx.com/>

[19] Widgets and Gadgets for Houdini. Accessed on July, 2007.

<http://www.houdinitools.com/>

[20] Houdini Development Kit. Accessed on August, 2007.

<http://odforce.net/>

[21] Stable Fluids in Java. Accessed on July, 2007.

<http://www.multires.caltech.edu/teaching/demos/java/stablefluids.htm>

[22] SPH-Based Fluid Simulation for Special Effects. Accessed on June, 2007.

<http://www.cescg.org/CESCG-2007/papers/>

[23] Fluid flow for the rest of us. Accessed on August, 2007.

<http://poseidon.cs.byu.edu/~cline/fluidFlowForTheRestOfUs.pdf>

[24] Jos Stam. Accessed on September, 2007.

http://en.wikipedia.org/wiki/Jos_Stam.

[25] Going with the Flowline. Accessed on September, 2007.

<http://vfxworld.com/>

[26] The Poseidon. Accessed on September, 2007.

<http://www.moving-picture.com/poseidonThe>