2D Stable Fluids

Yunfei Ouyang
University of Toronto
Toronto, Ontario, Canada
yunfei.ouyang@mail.utoronto.ca

ABSTRACT

In physics, fluid dynamics is a subdispline of fluid mechanics that describe the motion of fluids, such as liquids and gases. Back in 1980s, fluid simulation was very expensive, and cost heavy. In 1999, Jos Stam published a highly influenced paper about fluid dynamics, called stable fluids. It proposed a stable solver to the incompressible Navier–Stokes equations in real time. Showing how fluid simulation can be efficient and simple. This paper will go through the steps of the stable fluids solver by Jos Stam and implement it in a interactable 2D grid.

CCS CONCEPTS

• **I.3.7 [Computer Graphics]: Two-Dimensional Graphics and Realism—Animation**;

KEYWORDS

Animation of fluids, Navier-Stokes, Stable solvers, Interactive, Gaseous phenomena, Physical Based Simulation

ACM Reference Format:
Yunfei Ouyang. 2021. 2D Stable Fluids. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Fluids such as Liquids and Gases are the core simulation area in computer graphics. Nowadays, movies and films all come with the realistic simulation of fires, explosions, atmosphere, etc., where the fluids solver are embedded inside the animation tools animator used. It seems very common that we can access tools like this in almost any 3d software, but it wasn't the case back then. Back in 1986, computational fluid dynamics still requires access to supercomputing facilities due to the non-linearity property of the Navier Stokes equation. This is where stable fluids come in, in 1999 SIGGRAPH, Jos Stam published a paper about stable fluids solver that is a pioneer in the computer graphic domain. The algorithms proposed by Jos Stam can run in real-time with arbitrary large time step, diffusion rate, and viscosity rate made the algorithm adopted by many animation software. It is still at the root of many simulation algorithms.

In this report, we will implement the Stable Fluids solver in a 2D grid, where the solver is mainly used to simulate gaseous-like phenomena. Instead of particle-based simulation, Jos Stam proposed a grid-based simulation which is very intuitive and visually pleasing. We first initialize a 2D grid where each grid cell is representing a density and velocity which are used by the solver. The basic structure of the solver is to first add force to the initial state of the grid, then solve the incompressible Navier Stokes equation in a piece by piece fashion, i.e. we take care of the adding force, advection, diffusion, projection separately. In the fixed grid cells, the density for each cell represents the number of imaginary particles in that cell. where we add force by the interactive system implemented in the project. The advection steps backtrack the grid cell in time to get the accurate "particle" update of the current cell. diffusion step diffuses the fluids by the given diffusion rate, and last, the projection step uses the famous Helmholtz-Hodge Decomposition to decompose a flow field into its divergence-free and curl-free field. It lies at the heart of fluid analysis.

2 RELATED WORK

Most previous publications in computer graphics before the stable fluids [1] were driven by visual appearance than its physical accuracy. 2D simulations were used by many of the methods due to the simplicity in computational space. Where the incompressible Navier Stokes equation Poisson solver was first implemented by Yaeger and Upson in the 1986 paper [5]. Later in 1987 Foster and Metaxas implemented the full 3D Navier Stokes equations [6]. Moreover, after the publication of the stable fluids, Jos Stam published more papers on the specific area of fluids simulation, he explained in detail the fast Fourier Transform methods in the stable fluids [3]. As well as a 3D visualization of smokes with Ronald and Henrik[2]. Moreover, a real-time fluid dynamic for games based on the Stable fluids [4].

3 METHOD

The 2D Stable Fluids is implemented with Python and used NumPy and Pygame library. Python is chosen here over C++ due to the easier debug routine. Moreover, 3D visualization is very complex and deserves another paper [2]/project. fast Fourier Transform on the other hand is directly provided by NumPy.fft and SciPy.fft, which means that the implementation will be simpler than the traditional routine (without the FFT). Therefore the project implements the 2D Stable Fluids Project with the traditional routine described in [1, 4].

3.1 The Navier Stokes Equation

Given spatial coordinate $X = (x, y)$, velocity field $\mathbf{u}$ and pressure field $\mathbf{p}$, with initial time $t = 0$, the Navier-Stokes equation describes fluids are given as follow [7]:

$$\nabla \cdot \mathbf{v} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla \mathbf{p} + v\nabla^2 \mathbf{u} + \mathbf{f} \tag{2}$$

Where v is the kinematic viscosity, ρ is the density and $\mathbf{f}$ is an external force. Next, the Helmholtz-Hodge Decomposition shows that for an arbitrary field $\mathbf{w}$, it can be decomposed into the following form:

$$\mathbf{w} = \mathbf{u} + \nabla q \tag{3}$$

Note that $\nabla \cdot \mathbf{u} = 0$. Defining an operator $\mathbf{P} = \nabla$, we have:

$$\nabla \cdot \mathbf{w} = \nabla^2 q \tag{4}$$

Moreover, we can apply the project on (2):

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}(-(\mathbf{u} \cdot \nabla)\mathbf{u} + v\nabla^2 \mathbf{u} + \mathbf{f}) \tag{5}$$

This is the equation we are using to implement the stable fluids solver.

3.2 Density Field

Since we are doing a 2D Grid fluid simulation, there are no particles involved, but we need a way to visualize the motion of the flow, this is where density field comes in handy. Let $\mathbf{s}$ be a density field of the 2D grid and $\mathbf{s}_{ij}$ represents the ij-th cell density. $\mathbf{s}_{ij}$ can be interpreted as how dense that cell is, i.e. denser $\longrightarrow$ more particles are in that cell. For an arbitrary scalar field $\mathbf{a}$ in the fluid, the update of $\mathbf{a}$ can be described similarly as (5):

$$\frac{\partial \mathbf{a}}{\partial t} = \mathbf{P}(-(\mathbf{u} \cdot \nabla)\mathbf{a} + k\nabla^2 \mathbf{u} - \alpha \mathbf{a} + \mathrm{S}) \tag{6}$$

Where k is a diffusion constant, α is a dissipation rate and S is a source.

4 STABLE FLUIDS

4.1 Grid Representation

To set up the solver, we decided to use a 2D grid, where given length n, our grid will have size $(n+2) \times (n+2)$, this is because to satisfy boundary conditions, we need two extra rows/cols to redirect the velocity and the density of the fluid. This setup allows a fixed boundary condition for the fluids. The grid will act like a box where fluids can not "leak out". Another boundary condition proposed in stable fluids [1] was a periodic boundary condition, which means that when fluids reach either side of the grid, instead of redirecting the density and the velocity of the fluid, we simply let pass the boundary to the other side like Pacman. Or more specifically, the grid will become a 2-dimensional torus.

4.2 Solver

To understand how the stable fluids solver works, we need to take a deeper look at equations (5) and (6). To avoid redundancy, we will use equation (5) since (6) follows after seeing how (5) updates. Equation (5) shows the update routine for an arbitrary fluid. Break it down we have four parts:
1. add force: $\mathbf{f}$
2. diffusion: $v\nabla^2 \mathbf{u}$
3. advection: $(\mathbf{u} \cdot \nabla)\mathbf{u}$
4. projection: $\mathbf{P}$
We will explain the four processes one by one.

4.3 Add Force

Because we are implementing an interactive fluid simulation, we need a way to add force from the user's mouse motion. A mouse motion has two kinds, one is a mouse click, another is mouse drag. We will consider mouse click as adding a source to the scalar field $\mathbf{a}$, in our case, adding density to the specific cell in the density field. Next, we consider a mouse drag as an increase in velocity of the X component and Y component, we will simply add velocity to the initial cell the mouse dragged in the velocity field of the fluid.

4.4 Diffusion

The next part is diffusion, note that diffusion doesn't depend on the velocity of the fluid, where it is the net movement of anything generally from a region of higher concentration to a region of lower concentration. In our 2D grid, we consider the grid cells separately, where each cell will be updated by becoming the average of the grid cells surrounding it, i.e. we take the average of the top, left, right and bottom cells with respect to this cell. Since the solver needs to be stable, basic linear interpolation is not stable when the diffusion rate is too large, this will result in unstable changes. Therefore we need to do it iterative such that:

$$s_t = s(x+1,y) + s(x-1,y) + s(x,y+1) + s(x,y-1)$$

$$s(x,y) = \frac{s_0(x,y) + k \cdot s_t}{1+4k}$$

Where s is the current scalar field, s_0 is the previous time step scalar field, and k is the diffusion rate. This prevents the overshoots of the diffusion update. The iterative method is so-called the Gauss-Seidel relaxation method. It can obtain a good result with very small iterations [1].

4.5 Advection

Advection is the transport of a substance or quantity by the motions of a fluid. Note that since we want to know the flow of the substances in the grid, it is very hard to update cells forward in time since the particles will not always lie in the center of a cell. Instead, we can track those particles that lie exactly in the center of a cell, then trace them back to find out the updated quantity. Linear interpolation is used to calculate the previous 2×2 grids density to the new density. The linear interpolation step is defined as follow:

$$s(x,y) = lp(lp(g_{10}, g_{11}, i_x), lp(g_{00}, g_{01}, ix)i_y)$$

Where s is the current substance, g_{10}, g_{11} are the bottom two grids, g_{00}, g_{01} are the top two grids. i_x and i_y are the ratio of the back traced substance location respected to the 2×2 grids. This finishes the advection step.

4.6 Projection

The projection method is an effective means of numerically solving time-dependent incompressible fluid-flow problems, it ensures that vector field is mass conserving. Note that this is achieved by using Hodge decomposition (3). A bit rearranging we then have:

$$\mathbf{u} = \mathbf{P}\mathbf{w} = \mathbf{w} - \nabla \mathbf{q} \tag{7}$$

Note that $\mathbf{u}$ is the divergence-free field, where we want to get it by subtracting $\mathbf{q}$ curl-free field from the vector field $\mathbf{w}$. Mathematically, we have the following:

$$\nabla \cdot \mathbf{v}(x,y) = \frac{\mathbf{v}_x(x+1,y) - \mathbf{v}_x(x+1,y) + \mathbf{v}_y(x,y+1) - \mathbf{v}_y(x,y-1)}{2}$$

Then we can use the Gauss-seidel method mentioned in diffusion step to solve for $\mathbf{q}$. Then,

$$\nabla \mathbf{q}(\mathbf{x},\mathbf{y}) = (\frac{\mathbf{q}(x+1,y) - \mathbf{q}(x-1,y)}{2}, \frac{\mathbf{q}(x,y+1) - \mathbf{q}(x,y-1)}{2})$$

Last we can solve the Hodge decomposition. Thus finish the projection.

4.7 Update

We update the velocity field and the density field separately, where the velocity field depends on the viscosity constant, and the density field depends on the diffusion rate. Here is the step for updating velocity field: diffuse($\mathbf{v}_x$), diffuse($\mathbf{v}_y$), project, advect($\mathbf{v}_x$), advect($\mathbf{v}_y$), project. For updating density field: diffuse($\mathbf{d}$), advect($\mathbf{d}$), dissipation(α) Note that for dissipation we simply divide the density field by $(1 + dt * \alpha)$. This conclude the stable fluids solver.

5 RESULTS

The implemented 2D fluid simulation works in real-time under Pygame GUI, where when tuning the GUI, at size(48 x 48) with diffusion = 0.0001, viscosity = 0.0001, dissipation = 0.01 yields Gaseous phenomena. Users can interact with mouse drag or click to put in smokes which then will be simulated by the solver. By pressing the "s" button the simulation will start, pressing the "r" button will reset the simulation, furthermore by pressing the "a" button the solver will be paused so the user can enjoy the beauty of fluids.

6 CONCLUSION

The implementation is complete, and there is a lot of places it can be improved on. Due to the time constrain I choose to implement the Solver under python, which is very inefficient compare to the precompiled programming language C++. Moreover, several aspects of the solver can be improved, for example, 3D grid stable fluids with fast Fourier Transform step to increase the real-time performance. Last, this is a very pleasant journey, thanks to all the supports from the class, and the Navier Stokes Equation is indeed very interesting!

7 REFERENCES

[1] J. Stam, Stable Fluids, In SIGGRAPH 99 Conference Proceedings, Annual Conference Series, August 1999, 121-128.

[2] R. Fedkiw, J. Stam and H. W. Jensen, Visual Simulation of Smoke, In SIGGRAPH 2001 Conference Proceedings, Annual Conference Series, August 2001, 15-22

[3] J. Stam, A Simple Fluid Solver based on the FFT, Journal of Graphics Tools Volume 6, Number 2, 2001, 43-52.

[4] Jos Stam "Real-time Fluid Dynamics for Games" Proceedings of the Game Developers' Conference, 2003

[5] L. Yaeger and C. Upson. Combining Physical and Visual Simulation. Creation of the Planet Jupiter for the Film 2010. ACM Computer Graphics (SIGGRAPH '86), 20(4):85–93, August 1986.

[6] N. Foster and D. Metaxas. Modeling the Motion of a Hot, Turbulent Gas. In Computer Graphics Proceedings, Annual Conference Series, 1997, pages 181–188, August 1997.

[7] A. J. Chorin and J. E. Marsden. A Mathematical Introduction to Fluid Mechanics. Springer-Verlag. Texts in Applied Mathematics 4. Second Edition., New York, 1990.