A Simple Method for Two-Dimensional Fluid Simulation

Kavan Lam
University of Toronto
kavan.lam@mail.utoronto.ca

ABSTRACT

This paper outlines a simple, but powerful algorithm for the simulation of fluids in two-dimensions. The algorithm that will be described in detail is entirely based on the algorithm described in the paper, "Real-Time Fluid Dynamics for Games" by Jos Stam [Stam 2003]. Although we follow this paper very closely, additional analysis will be provided to give readers a deeper understanding of the material presented in [Stam 2003] and readers can optionally read the appendix section attached to this paper which includes handwritten derivations. The paper will begin with a brief introduction to the world of fluid simulation and the Navier-Stokes equations for incompressible fluids. Then, the implementation and results will be discussed in detail.

1 INTRODUCTION

The simulation of the flow and behavior of fluids both in two and three dimensions have been worked on by hundreds if not thousands of researchers and engineers over the past decade. This is clear from the abundance of papers, resources and projects created over the years relating to fluid simulation. The simulation of fluids itself has many applications across a wide range of fields and you yourself have probably already seen it in action simply by watching movies and/or playing video games over the years.

Now, when it comes to the actual implementation of fluid simulators, it is almost always based on the famously known Navier-Stokes equation which can be used to model any fluid. In conjunction with the Navier-Stokes equation is the famously known, divergence free constraint which holds a crucial role in many fluid simulators [Levin 2020]. Below are the two equations.

$$\frac{\partial \vec{U}}{\partial t} = -\vec{U} \cdot \nabla \vec{U} - \frac{1}{\rho} \nabla P + V \nabla^2 \vec{U} + \vec{F}$$

$$\nabla \cdot \vec{U} = 0 \ (divergence\ free)$$

In the equation above, $\vec{U}$ is the velocity, ρ is the density, P is the pressure and $\vec{F}$ is the sum of external forces.

To be clear, the usage of the divergence free constraint with respect to fluid simulation, simply means the fluid is incompressible and this is a great constraint for simulating fluids such as water which is very hard to compress. For our fluid simulator, we will also be using the divergence free constraint. Note that the constraint may also be referred to as the conservation of mass in the context of fluid simulation.

To conclude the introduction, lets discuss what exactly we are trying to accomplish and how. The paper "Real-Time Fluid Dynamics for Games" by Jos Stam [Stam 2003] is an amazing paper which provides an elegant algorithm for simulating two- or three-dimensional fluids. Our goal is to implement a two-dimensional simulation of incompressible fluids by following as closely as possible the algorithm proposed by Stam.

2 METHOD

In this section we will break down the algorithm proposed in [Stam 2003] and describe our exact implementation which has slight modifications. The first thing to iron out is what the equations of motion will be as these will dictate what our future algorithmic steps are. In [Stam 2003], there are two main equations used, the previously mentioned Navier-Stokes equation and the equation which models the flow of some density present in some velocity field. The Navier-Stokes equation listed in the paper is however missing the term with the pressure, which we are speculating is mixed in with the external force term. Anyhow, we add that term back in as it will become relevant for the pressure projection step. At the end of the day our equations used to model the problem is as follows

$$\frac{\partial \vec{U}}{\partial t} = -\vec{U} \cdot \nabla \vec{U} - \frac{1}{\rho} \nabla P + V \nabla^2 \vec{U} + \vec{F} \quad (1)$$

$$\nabla \cdot \vec{U} = 0 \ (divergence\ free) \quad (2)$$

$$\frac{\partial \rho}{\partial t} = -(\vec{U} \cdot \nabla)\rho + k \nabla^2 \rho + S \quad (3)$$

where k is the diffusion rate and S are the sources of density. The usage of density here is a bit misleading and the correct way to understand density in this implementation is to think of it as the amount of dye. Suppose you have completely black water that completely fills your 2D view of the fluid. From a 2D viewpoint, if the black water moves you will not be able to see the motion as everything is just black, but if you add some white dye into the black water then you will see the dye mixing and flowing about in the black water. For our simulation, the black water acts as our velocity field which is controlled by (1) and (2) and our dye (aka density) is controlled by (3). So really the fluid we are simulating is the dye and not the black water. Now, using these equations we can formulate our update algorithm for advancing the fluid forward in time which is where the bulk of the work gets done. More generally here is the high-level idea of the full algorithm.

1) Initialize all our data structures used to hold the state of our 2D fluid system
2) Advance our fluid forward in time
3) Render the fluid using Processing (Java library)
4) Detect user input to add dye and change the velocity field
5) Repeat steps 2 to 4 until simulation is closed

Steps 1, 3 and 4 may vary between different implementations and is not important enough to discuss in this paper and so we focus all our attention to step 2. Advancing the fluid forward in time requires the following steps.

1) Diffusion of the x and y component of the velocity
2) Pressure Project applied to the velocity field

3) Advection of the x and y components of the velocity
4) Pressure Projection applied to the velocity field again
5) Diffusion applied to the dye (density)
6) Advection applied to the dye (density)

The steps above can be broken down into three categories which are Diffusion, Advection and Pressure Projection (in [Stam 2003] it is simply called Projection). Each one of these steps can be traced back to the equations of motion stated at the beginning of this section, but we will not go into the details of that to keep this paper short. For instance, the terms with the Laplacian operator is the reason for the diffusion steps. It is also important to note that in each of the 6 steps we are considering boundary conditions. The resolution for the boundaries in our implementation simply sets the values at boundary cells equal to its immediate neighbour for the dye and the negative of its immediate neighbour for velocities. Corner cells are resolved by taking the average of the two immediate neighbours.

2.1 Diffusion

You can think of diffusion here as the natural tendency for some physical quantity to want to spread out such that the concentration of said quantity is equally distributed. For example, in step 5 where the diffusion sub routine is applied to the dye, this is what allows the dye to spread out and "dissolve" even in the presence of a zero-velocity field. In our implementation the sub routine is given a diffusion rate which determines how quickly the dye or even the velocity itself diffuses into its neighbours. In practice what this means is going through the all the cells (one cell is one pixel and it discretizes our fluid) and computing a new value of the quantity in question. The update equation for each cell from [Stam 2003] is as follows.

$$X(i,j) = (X_0(i,j) + a(X(i-1,j) + X(i+1,j) + X(i,j-1) + X(i,j+1)))/(1+4*a)$$

where X is the new values, X_0 are the original values and a is a diffusion rate. There is one update equation like the one above for each cell and therefore a linear system can be generated and solved. For solving, the Gauss-Seidel method is used, and you can refer to appendix B for more information. The usage of Gauss-Seidel means we must iterate over all the cells multiple time and in [Stam 2003] 20 iterations are used.

2.2 Advection

Advection is the bulk movement of some quantity and in our case would be the dye and the velocities. Advection applied to the dye makes sense as the dye itself will get translated spatially over time due to the velocity field and this is what allows you to see the dye swirling around. However, the advection applied to the velocities themselves is not as sensible and in [Stam 2003], Stam calls this self-advection. Either way the advection sub routine is implemented the same way whether we are advecting the dye or velocities themselves. The actual implementation is quite simple and only requires us to go through each cell and compute a new value for the quantity in question based on the current values and the velocity field. To be specific, for each cell C, we use the current velocity field to compute which cell c, would have moved to the one we are on currently and use the value from c and its immediate neighbours to compute the new value for cell C. For non-integer cell positions we linear interpolate the values. Finding cell c is as simple as taking the velocity at cell C and moving in the opposite direction with some provided time step.

2.3 Pressure Projection

The pressure projection step is responsible for updating the velocity field such that it satisfies the divergence free constraint. It involves solving a Poisson equation as mention by Stam [Stam 2003]. The exact Poisson equation is the same one mentioned in [Levin 2020]. In general, the projection step as proposed by Stam is as follows.

1) Compute the divergence of the velocity field via first order approximations
2) Compute the pressures by solving a linear system which relates the divergence of the velocity field to the pressures (Gauss-Seidel is used to solve the linear system)
3) Compute the updated values for the velocities using the pressures

The exact reason why these are the steps and why it works can be found in appendix A. The update equation for a single cell in step 2 according to [Stam 2003] is

$$P(i,j) = (div(i,j) + P(i-1,j) + P(i+1,j) + P(i,j-1) + P(i,j+1))/4$$

and in step 3 the update equations per cell is as follows.

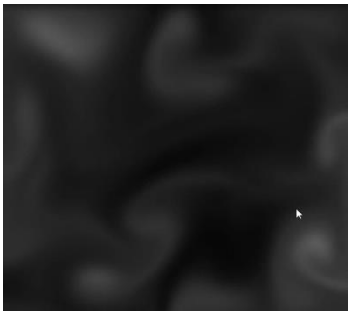
$$U_x(i,j) \mathrel{-}= 0.5 * (P(i+1,j) - P(i-1,j))/h$$

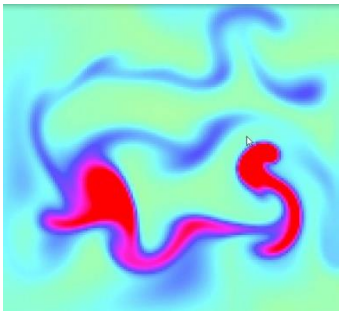
$$U_Y(i,j) \mathrel{-}= 0.5 * (P(i,j+1) - P(i,j-1))/h$$

In the equations above, h is the spacing between the particles that make up the fluid and the exact value is up to the implementor. In our case, we also multiply the equations in step 3 by the time step and divide by density and you can read why in Appendix A.

3 RESULTS and SUMMARY

Overall, the results that we got are great and is quite performant (at least in 2D). Below are two screenshots.





In order obtain a better understanding of the results, you are recommended to watch the video here https://youtu.be/A617n6Sj7XM.

A natural extension to this project is to implement it in 3D and this is possible using the same algorithm. The algorithm described in [Stam 2003] does not restrict us to 2D and an implementation in 3D would not require significantly more implementation time. However, introducing the third dimension will increase the computational complexity from $O(n^2)$ to $O(n^3)$ and a more sophisticated process will be required to render the fluid. So, although the third dimension is implementable, it is questionable whether the simulation will remain performant and perhaps the vectorization of the computations may be necessary.

REFERENCES

Jos Stam. 2003. Real-Time Fluid Dynamics for Games. https://www.researchgate.net/publication/2560062_Real-Time_Fluid_Dynamics_for_Games.

David Levin. 2020. Physics-based animation lecture 10: Fluid Simulation. https://www.youtube.com/watch?v=VddQZH_Ppd0&feature=youtu.be

Appendix A

CSC 417: Final project
KAVAN LAM (1003038802)
Dec 21, 2020

The following will gives readers a better understanding of the pressure projection and algorithm implemented. The paper by Jos Stam uses this process but does not fully explain why which is what this appendix will do.

The pressure projection or simply the projection step can be broken down into 3 main steps.

1) compute the divergence using the x and y components of the velocity field
2) compute the pressures by solving a linear system involving the divergence
3) compute the updated values for the velocities using the pressures

Why are these the steps? what exactly is the linear system from step 2? we will see right now and we begin with the Navier-Stokes equations for incompressible fluids (which are the type of fluids we are simulating)

$$\frac{\partial \vec{u}}{\partial t} = -\vec{u}\cdot\nabla\vec{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u} + \vec{F} \quad \text{and} \quad \nabla\cdot\vec{u} = 0$$

For the pressure projection we focus on $\boxed{\frac{\partial \vec{u}}{\partial t} = -\frac{1}{\rho}\nabla p}$ where ρ = density and p = pressure

(Remember that we treat each term separately)

Now we know that the velocity $\vec{v}$ can be broken down into the x and y components and we can also use a first order approximation on the time derivative on the left side additionally we can use central differences for approximating ∇p. Doing all of this results in the two following equations.

$$① \quad \vec{v}_x(x,y,T+\Delta T) = \vec{v}_x(x,y,T) - \frac{\Delta T}{\rho} \frac{P(x+h,y,T) - P(x-h,y,T)}{2h}$$

$$② \quad \vec{v}_y(x,y,T+\Delta T) = \vec{v}_y(x,y,T) - \frac{\Delta T}{\rho} \frac{P(x,y+h,T) - P(x,y-h,T)}{2h}$$

Now we look at $\nabla \cdot \vec{v} = 0$ (conservation of mass / incompressibility constraint)

$$\nabla \cdot \vec{v} = \frac{\partial \vec{v}_x}{\partial x} + \frac{\partial \vec{v}_y}{\partial y} = 0 \quad \text{(we can again use central differences)}$$

$$\Rightarrow \frac{\vec{v}_x(x+h,y,T+\Delta T) - \vec{v}_x(x-h,y,T+\Delta T)}{2h} + \frac{\vec{v}_y(x,y+h,T+\Delta T) - \vec{v}_y(x,y-h,T+\Delta T)}{2h} = 0$$

Now subbing in what we have for ① and ② we get the following

$$\left(\frac{\left(\vec{v}_x(x+h,y,T) - \frac{\Delta T}{\rho}\frac{P(x+2h,y,T) - P(x,y,T)}{2h}\right) - \left(\vec{v}_x(x-h,y,T) - \frac{\Delta T}{\rho}\frac{P(x,y,T) - P(x-2h,y,T)}{2h}\right)}{2h} + \frac{\left(\vec{v}_y(x,y+h,T) - \frac{\Delta T}{\rho}\frac{P(x,y+2h,T) - P(x,y,T)}{2h}\right) - \left(\vec{v}_y(x,y-h,T) - \frac{\Delta T}{\rho}\frac{P(x,y,T) - P(x,y-2h,T)}{2h}\right)}{2h} \right) = 0$$

Now we collect all the velocity terms on the left and pressure terms on the right

$$\vec{v}_x(x+h,y,T) - \vec{v}_x(x-h,y,T) + \vec{v}_y(x,y+h,T) - \vec{v}_y(x,y-h,T) = \frac{\Delta T}{2h\rho}\Big[\left(P(x+2h,y,T) - P(x,y,T)\right) - \left(P(x,y,T) - P(x-2h,y,T)\right) + \left(P(x,y+2h,T) - P(x,y,T)\right) - \left(P(x,y,T) - P(x,y-2h,T)\right)\Big]$$

$$\frac{2h\rho}{\Delta T}\left(\vec{v}_x(x+h,y,T) - \vec{v}_x(x-h,y,T) + \vec{v}_y(x,y+h,T) - \vec{v}_y(x,y-h,T)\right) = -4P(x,y,T) + P(x+2h,y,T) + P(x-2h,y,T) + P(x,y+2h,T) + P(x,y-2h,T)$$

we see that the left side is just the scaled divergence of the velocity field and we call this whole left side $DIV(x,y)$. The right side is a bunch of unknown pressure values and thus we have ourselves a linear system. we can solve via the Gauss-Seidel method. To use the method we isolate for $P(x,y,T)$ so we get...

$$P(x,y,T) = \frac{-D(x,y) + P(x+2h,y,T) + P(x-2h,y,T) + P(x,y+2h,T) + P(x,y-2h,T)}{4}$$

which resembels what is used in the code.

Once the pressures has been computed we can use ① and ② as our update equation for the velocities. This concludes the reasoning for the algorithm proposed by Jos Stam (which is used in our implementation)

Apperdix B

The following gives the details of the Gauss-seidel Method which is used extensively in our 2D fluid simulation. Simply, the Gauss-seidel method is an iterative method for approximating the solution of a system of n linear equations in n variables where $n \in \mathbb{N}$.

Assume you have the following linear system ...

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ \vdots \quad \vdots \quad & \quad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

(a_{nn} and b_n are known constants; x_n are the unknowns)

The method requires that we isolate the n^{th} equation for the n^{th} unknown so...

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{11}x_1 - a_{12}x_2 - \cdots - a_{1n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1})$$

we then make an initial guess for the unknowns (ie/assigne a value to each x_n). Finally we update our guess for the unknowns by evaluating the right side of the equations above using our current guess.

If the system converges then as the number of iterations goes to infinity the guess for the unknowns reach the actual solution. The number of iterations here refer to how many times we need to update / recompute our guess.

The paper by Jos Stam uses 20 iterations.