Fluid flow and the Kelvin–Helmholtz instability

Tom Rindell

June 17, 2025

Abstract

This project works on an implementation of fluid flow simulation method developed by Jos Stam [1] in order to study Kelvin-Helmholtz instability. The simulated fluid is wrapped around both in x and y borders resulting in torus topology. In order to visualize the flow of the fluid, some parts of the fluid are dyed and the dissolution of the dye in the flowing fluid is rendered in real-time on screen. The simulation will demonstrate formation of vortices at the boundary of two regions with different velocity. All of the code in this project is written in C++.

1 Introduction

At a boundary of two adjacent sections of a fluid with sufficiently different velocities, the fluid is very susceptible to small disturbances which result in a turbulent flow. This is known as Kelvin-Helmholtz (KH) instability [4]. The purpose of this project is to simulate fluid flow with KH-instability.

Fluid flow simulations tend to be computationally expensive, making them unfeasible for real-time rendering. In 1999 Joe Stam introduced a method for fluid simulation [2] which manages to alleviate this problem. It is relatively simple to implement, however it provides a notable improvement in the stability and computational efficiency for the fluid simulation over contemporary methods. The drawback is a decrease in the physical accuracy of the simulation. Still, for the purposes of real-time fluid flow rendering this is a reasonable trade-off as the fluid flow does – despite small physical inaccuracy – look visually convincing. More importantly, for the purposes of this project the physical accuracy is sufficient enough for demonstrating KH-instability.

2 Methods

Evolution of an incompressible fluid can be universally described with Navier-Stokes equations

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

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

where $\mathbf{u}$ is a vector field corresponding to the velocity of the fluid at a given point, ρ, p, and ν are the density, pressure, and viscosity of the fluid respectively, and $\mathbf{f}$ is the external force applied on the fluid. For an incompressible fluid $\partial_t \rho = 0$ and therefore the equation (1) functions as a continuity equation for constant density.

As the density, viscosity, and the external force are independent variables, the only term in equation (2) which can impose the incompressibility condition of equation (1) is ∇p. The incompressibility can be imposed on the equation (2) – and thereby ridding us of the pressure

term – through introduction of Helmholtz-Dodge decomposition $\mathbf{w} = \mathbf{u} + \nabla q$, where $\mathbf{u}$ satisfies the condition of equation (1) and q is some scalar field. Later, it'll be shown how the ∇q term is acquired from the divergence $\nabla \cdot \mathbf{w} = \nabla^2 q$. Now we can define a projection operator $\mathbf{P}$ which projects $\mathbf{w}$ to its divergence-free part $\mathbf{u}$ as

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

Operating with $\mathbf{P}$ on both sides of equation (2) we have

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} + \mathbf{f}\right), \tag{3}$$

where $\mathbf{P}(\nabla \mathbf{p}) = 0$. This formulation of the Navier-Stokes equations will be the one used for Stam's fluid simulation. The variables $\mathbf{u}$ and $\mathbf{f}$ are discretized for a numerical simulation and thereby defined on a $N \times N$ grid. Additionally, we impose periodic boundary conditions on the fluid by wrapping the x and y-coordinates. This torus topology will allow us later to use FFT in steps 3 and 4 of the algorithm.

For every time-step Δt, each term on the right-hand side of equation (3) us added one at a time after which the projection operation $\mathbf{P}$ is performed. Hence, Stam's simulator proceeds in four steps: addition of force, advection, diffusion, and projection. The velocity values are updated for every point in the grid before proceeding to the next step.

The first step – the addition of force – is straight-forward:

$$\mathbf{w}_1(\mathbf{x}) = \mathbf{w}_0(\mathbf{x}) + \mathbf{f}(\mathbf{x}, t)\Delta t.$$

This is a reasonable approximation if there if no significant time variation in $\mathbf{f}$ during the time step Δt.

The second step applies the advection for which the equation

$$\frac{\partial \mathbf{w}_2}{\partial t} + (\mathbf{w}_2 \cdot \nabla)\mathbf{w}_2 = 0 \tag{4}$$

has to be solved. This can be done with method of characteristics. A more detailed approach can be found in Stam's paper [2], however the method can be briefly understood as follows:[1] Solving for x and y values separately, let us denote one-dimensional component of $\mathbf{w}_2$ with just w_2 as the method is equivalent for both x and y components. The equation (4) can then be taken to be the total derivative

$$\frac{\mathrm{d}w_2}{\mathrm{d}t} = \frac{\partial w_2}{\partial t} + (\mathbf{w}_2 \cdot \nabla)w_2 = 0, \tag{5}$$

if $\mathbf{w}_2$ is interpreted as the derivative of the position vector $\mathbf{x}$. With this interpretation the equation (5) represents the flow of conserved velocity component w_2 in the fluid. For a small time-step Δt the value for w_2 can therefore be "traced back" by taking the value from $\mathbf{x} - \mathbf{w}_1\Delta t$. Taking both the x and y components together, the solution is then

$$\mathbf{w}_2(\mathbf{x}) = \mathbf{w}_1(\mathbf{x} - \mathbf{w}_1\Delta t)).$$

For the diffusion term we have the discretization $\Delta \mathbf{w}_3 = \nu\Delta t\nabla^2\mathbf{w}_3$, giving us

$$(\mathbf{I} - \nu\Delta t\nabla^2)\mathbf{w}_3(\mathbf{x}) = \mathbf{w}_2(\mathbf{x}), \tag{6}$$

[1] I found this step to be a bit more difficult to understand than the others and therefore my explanation here might be a bit convoluted.

which we shall solve with Fourier transform. The choice of periodic boundary conditions makes application of FFT here convenient. Once performed, in the Fourier domain the nabla operator ∇ becomes ik transforming (6) into

$$\mathbf{w}_3(\mathbf{k}) = \frac{\mathbf{w}_2(\mathbf{k})}{1 + \nu \Delta t k^2}.$$

The final projection step is performed while still in the Fourier domain. The goal is to solve ∇q from the divergence of $\mathbf{w}_3$ and subtract that value from $\mathbf{w}_3$ as follows:

$$\nabla \cdot \mathbf{w}_3 = \nabla^2 q, \qquad \mathbf{w}_4 = \mathbf{w}_3 - \nabla q.$$

In the Fourier domain this simplifies to

$$\mathbf{w}_4(\mathbf{k}) = \mathbf{w}_3 - \frac{\mathbf{k}}{k^2}(\mathbf{k} \cdot \mathbf{w}_3(\mathbf{k}))$$

For visualization of the fluid, a dye field D is introduced which satisfies the advection equation

$$\frac{\partial D}{\partial t} + (\nabla \cdot \mathbf{u})D = 0.$$

This is solved the same way as for the fluid itself in the step 2 of the simulation algorithm.

3 Implementation

Four $N \times (N + 2)$ arrays are allocated. The two additional columns are required by the FFT function. Arrays `u`, and `v` store the initial values at the beginning of an iteration whereas the updated values are stored in the arrays `u0`, and `v0`. On the next iteration the roles are swapped; what was the second set of grids now functions as the set of initial values and the updated values are stored into the first set of grids.

Fluid flow solver used here is a slightly altered version of the code from Stam's paper [1]. In Stam's version of the algorithm the x and y-directional forces $\mathbf{f}_x$ and $\mathbf{f}_y$ are given as the function parameters `u0` and `v0` for the `stable_solve` function in which those variables are then overwritten to store the updated values for $\mathbf{u}_x$ and $\mathbf{u}_y$. I found this implementation to be slightly inconvenient to work with as the values for the force field are lost due to the overwriting and therefore I altered the function to take in the values for $\mathbf{f}_x$ and $\mathbf{f}_y$ as additional function parameters `f_x` and `f_y` instead. Perhaps there would have been an efficient way to use the Stam's code as is but I couldn't come up with a way which did not involve convoluted work-arounds.

As the forces are often applied only for a finite number of steps in the beginning, in addition to defining two $N \times N$ arrays `f_x` and `f_y` for the x and y-directional forces I had also defined an empty array `empty`. Once the forces have been applied for the desired number of steps, `empty` is submitted to the `stable_solve` function instead of `f_x` and `f_y`. Even though this isn't the most memory-efficient way of handling this, for relatively small values of N this isn't a problem. The reason for doing it this way instead of reallocating the values for the force array was to avoid any kind of noticeable stuttering in the simulation at the time-step where application of force stops.

For visualization of the results I wanted to create real-time rendering of the fluid flow. For this I used SDL2 library[3] as I am somewhat familiar with it. I don't believe the details of SDL2 are particularly relevant for this project and therefore I will only cover it here briefly. A GUI window

is created with `SDL_Window` object and pixels for that window are drawn with `SDL_Rederer` object. At the beginning of each iteration the `render` function is called which goes over every pixel on the screen and assigns it a color based on the value of the dye field at the corresponding point. Once all of the points have been assigned a value, `SDL_RenderPresent` will refresh the window with updated values.

`src` directory contains a Makefile for compiling the code. Compilation can be done by running `make` in the `src` directory. This creates `fluid_flow` executable which can be run from the `run` directory.

4 Results

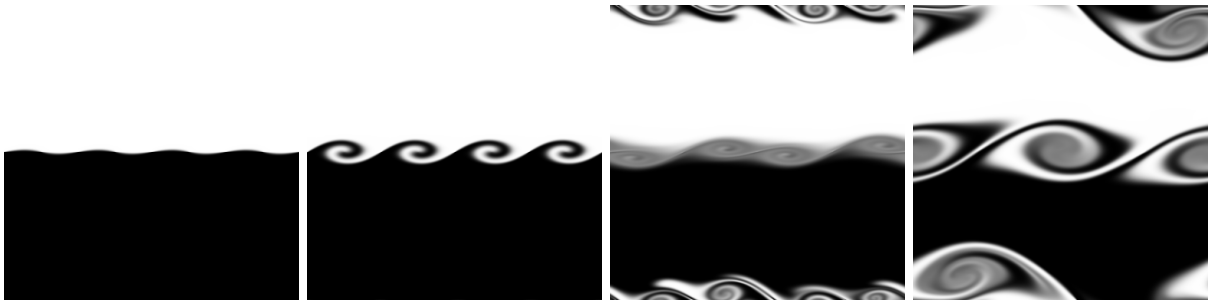

Figure 1: Evolution of the fluid flow after 20, 100, 300, and 500 simulation steps. Here white corresponds to the value where dye is at most intense whereas the black regions contain no dye at all.

Figure 1 shows the evolution of the fluid by applying the parameters given in the project instructions. The initial forces are defined in such a way that ...As can be seen from the figure, in the beginning at 20 simulation steps the dye is almost unperturbed. At 100 steps turbulent vortices are emerge. At 300th step the initial vortices have almost dissipated with new vortices emerging at the edge where y-boundary is wrapped. At 500th step new – and larger – vortices are emerging in the middle.

The nature of the flow can be changed significantly by adjusting the parameters For instance, by increasing the viscosity, the fluid will instead stabilize after an initial perturbation. Decreasing the viscosity on the other hand will tend to increase the instability as would be expected.

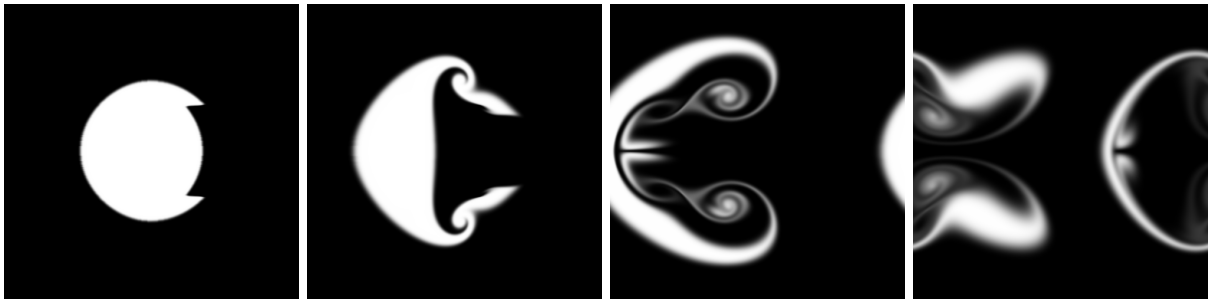

Figure 2: Evolution of the fluid flow 20, 100, 300, and 500 simulation steps. A leftward force positioned at the central right of the blob is applied for 10 simulation steps

I had also attempted to replicate the flowing blob of dye from the project instruction sheet. This turned out to be a bit more difficult than I expected but I am quite happy with the result which is shown in the figure 2.

For convenience, I implemented a function for applying a circular blob of dye by pressing the left mouse button. This allows for a convenient way of stydying the behavior of the fluid.

5 Conclusions

The application of Stam's simulator seems to have been successful. An initial force creates disturbances in the flow, resulting in occasional turbulent vortices and eventual mixing of the dye. A demonstration of Kelvin-Helmholtz instability was successful and I learned a lot about fluid simulations through this project.

The program could be extended to have a more streamlined method for interacting with the flow. For instance, an external force could be then applied by right-clicking and dragging the cursor. This would however be mostly a matter of convenience. Possible extension for three-dimensional simulator should be fairly straight-forward. Although an implementation of renderer for that three-dimensional fluid flow would probably be a nightmare.

References

[1] Jos Stam. 2001. A Simple Fluid Solver Based on the FFT. Journal of Graphics Tools, 6(2), 43–52. https://doi.org/10.1080/10867651.2001.10487540

[2] Jos Stam. 1999. Stable fluids. In Proceedings of the 26th annual conference on Computer graphics and interactive techniques (SIGGRAPH '99). ACM Press/Addison-Wesley Publishing Co., USA, 121–128. https://doi.org/10.1145/311535.311548

[3] SDL, GitHub repository, https://github.com/libsdl-org/SDL

[4] Kelvin–Helmholtz instability (2025, May 22). In Wikipedia. https://en.wikipedia.org/wiki/Kelvin%E2%80%93Helmholtz_instability