Stable Fluids

Victor Videau
victor.videau@polytechnique.edu

Irada Bunyatova
irada.bunyatova@polytechnique.edu

Section 1. Method Description

The paper of Jos Stam on Stable Fluids is based on grid representation, which allows us to represent the density of the fluid for each cell of the grid. According to the method being implemented, the values of velocity and density are defined in the center of the cells. The stable fluid model is based on the Navier Stokes equations considering that the fluid is mass conserving. This implementation allows us to observe fluid-like behavior in a reasonable amount of time, however it does not provide enough physical accuracy to build a physics model. The stable fluid simulation is based on the equation of velocity and density:

$$\frac{\partial u}{\partial t} = P(-(u \cdot \nabla)u + \nu \nabla^2 u + f) \qquad \frac{\partial a}{\partial t} = -u \cdot \nabla a + k \nabla^2 - \alpha a + S$$

Equation 1 and 2. Velocity and density equations

Solving the velocity equation

Instead of adding the values of velocities, each step in simulation is done by taking the value of velocity which was calculated at the previous step. The order of solving the velocity equation is the following: adding force, transporting, diffusing. Each of these three steps accounts for calculating a part of the equation of velocity (equation 1). To imply after these three steps that the fluid is divergence free, the projection step is being used.

Solving the density equation

We can note that the equations for density and velocity are similar. To find the value of density at each simulation we follow the same first three steps: adding force, transporting, diffusing. This is then followed by the dissipation step, which accounts for the term in the equation with dissipation rate α of the substance.

For this implementation two types of grids are used: one for velocity and another one for density. Both for velocity and density values in the current and previous time step should be stored. At each iteration of the simulation, the new values for the velocity and density are computed based on the values obtained at the previous iteration. To do so, the values of previous and current velocities / densities are swapped in the beginning of every simulation step. After this, the velocity equation is solved, which is followed by the density solving step.

Section 2. Details of Implementation

We implemented the simulation of the fluid for 2D. In our implementation the density is stored in a 2D array of float of size N x N. The velocity is stored in a 3D array of size 2 x N x N array, for the two dimensions of the velocity. We take square cells of length 1.

Boundaries

We introduced a function for handling the boundaries of the grid, since the calculations involving the 4 neighbors of the cell cannot be performed on boundaries. We decided to implement the fixed boundaries : the fluid should not exit the grid. When the direction of the normal velocity at boundary cells is opposed to the boundary, we keep the velocity in its original direction. Otherwise, we set it to minus its value.

Adding force

At the beginning of each time step, forces can be provided by the user interaction. We multiply these forces by the time step *dt* and we add them to the current velocity field.

Transport

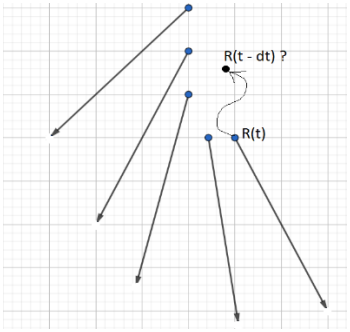
The second step of simulation corresponds to the advection of the fluid. This step accounts for the fluid movement according to the flow. In the Transport step, we use the velocity field to find the new position of the fluid particle. The idea is the following: given a fluid particle, we trace it back over a time step –dt to get its previous location. For the backtracking, we used a *second order Runge-Kutta* (RK2) method. The RK2 uses the slopes of two points to extrapolate the value at a time t + Δt. This method is used because it is robust for high time steps. Below is the RK2 formula for particle tracing (equation 3). We adapt it for the backtracking by taking Δt = -dt.

$$R^* = R(t) + V\big(R(t)\big) \triangle t$$

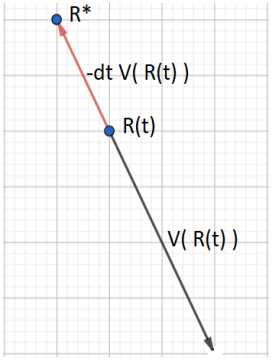
$$R(t + \triangle t) = R(t) + 0.5(V(R(t)) + V(R^*)) \triangle t$$

Equation 3. Runge Kutta 2 for particle tracing. D.E.Melnikov and V.M.Shevtsova(2005) Liquid Particles Tracing in Three-dimensional Buoyancy-driven Flows

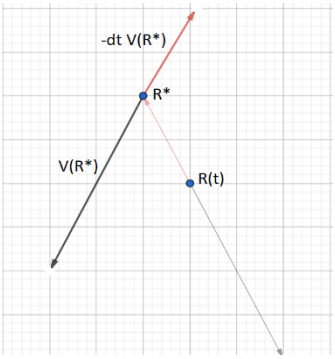
(1) In the above formula, the time step is noted by Δt. In our case it is –dt because we propagate backward in time. Given R(t) a point R at time t, the goal is to backtrace it to get R(t – dt) : its location at the previous time step. The point R corresponds to the fluid particle we want to backtrace.



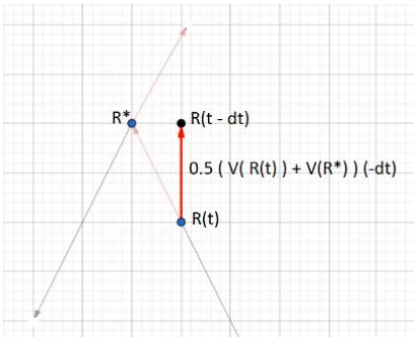
(2) V denotes the velocity, so V(R(t)) is the velocity of the particle R at time t. The first step is to compute the coordinates of a point R*. The first order Euler's formula is applied: the velocity at R(t) is multiplied by minus the time step and added to R(t). This gives us R*.



(3) The next step is to take the velocity at R* and to multiply it by minus the time step. This gives us the vector –dt V(R*).



(4) The final step is to take the average of the 2 vectors –dt V(R*) and –dt V(R(t)). This gives the vector in red. We add this vector to the original point R(t) and we obtain R(t – dt).



The point R(t – dt) will be most likely between some grid cells. To get the velocity at this point, we use a linear interpolation. Finally, we set the new velocity at R(t) at this value.

Diffuse

The diffusion step accounts for the effect of viscosity in the equation 1**.** In this step the goal is to solve the diffusion equation $\frac{\partial u}{\partial t} = \nu \nabla^2 u$. In our case it gives a sparse linear system Ax = b where b is a vector containing one dimension of the velocity field at every grid cell (without the boundaries), the solution x is the vector containing the velocity field in this one dimension at the next time step and A is a sparse matrix for which we had to derive the coefficients based on Jos Stam's Appendix B(). Having N x N grid, vectors b and x are of size (N-2) * (N-2) and A is a square matrix of side (N-2) * (N-2).

```
K1*(S[i-1,j,k]-2*S[i,j,k]+S[i+1,j,k]) +
K2*(S[i,j-1,k]-2*S[i,j,k]+S[i,j+1,k]) +
A[k]*S[i,j,k-1]+B[k]*S[i,j,k]+    .
```

Equation 4. System of finite difference equations to be solved

The end of the formula was missing in the paper and we deduced that it was **+ C[k] * S[i, j, k+1] = S0.** where S0 is the velocity at the previous time step.

```
K1 = -dt*k/(D[0]*D[0]),
K2 = -dt*k/(D[1]*D[1]),
A[k] = C[k] = -dt*k/(D[2]*D[2]) and
B[k] = 1+2*dt*k/(D[2]*D[2]),
```

Equation 5. Diffusion solver coefficients

Jos Stam gives the coefficients for the diffusion solver. D[0], D[1] and D[2] are respectively the size of the grid cells in the x, y and z axis. We show how we derived the coefficients to fill the sparse matrix A in our case.

By putting (1) and (2) together with D_n = D[n] for n = 0, 1, 2 :

$$S0 = -\frac{dt\,k}{D_0^2}\left(S_{i-1,j,k} - 2\,S_{i,j,k} + S_{i+1,j,k}\right) - \frac{dt\,k}{D_1^2}\left(S_{i,j-1,k} - 2\,S_{i,j,k} + S_{i,j+1,k}\right) - \frac{dt\,k}{D_2^2}S_{i,j,k-1} + \left(1 + 2\frac{dt\,k}{D_2^2}\right)S_{i,j,k} - \frac{dt\,k}{D_2^2}S_{i,j,k+1}$$

$$= -\frac{dt\,k}{D_0^2}\left(S_{i-1,j,k} + S_{i+1,j,k}\right) - \frac{dt\,k}{D_1^2}\left(S_{i,j-1,k} + S_{i,j+1,k}\right) - \frac{dt\,k}{D_2^2}\left(S_{i,j,k-1} + S_{i,j,k+1}\right) + \left(1 + 2\frac{dt\,k}{D_0^2} + 2\frac{dt\,k}{D_1^2} + 2\frac{dt\,k}{D_2^2}\right)S_{i,j,k}$$

$$= -\frac{dt\,k}{D_0^2}\left(S_{i-1,j,k} + S_{i+1,j,k}\right) - \frac{dt\,k}{D_1^2}\left(S_{i,j-1,k} + S_{i,j+1,k}\right) + \left(1 + 2\frac{dt\,k}{D_0^2} + 2\frac{dt\,k}{D_1^2}\right)S_{i,j,k} \quad \text{for the 2D case}$$

$$= -\frac{dt\,k}{D_0^2}\left(S_{i-1,j,k} + S_{i+1,j,k}\right) - \frac{dt\,k}{D_1^2}\left(S_{i,j-1,k} + S_{i,j+1,k}\right) + \left(1 + 2\frac{(D_0^2+D_1^2)\,dt\,k}{D_0^2\,D_1^2}\right)S_{i,j,k}$$

$$= -dt\,k\left(S_{i-1,j,k} + S_{i+1,j,k}\right) - \mathrm{dt\,k}\left(S_{i,j-1,k} + S_{i,j+1,k}\right) + (1 + 4\,dt\,k)\,S_{i,j,k} \quad \text{since } D_0 = D_1 = 1 \text{ in our case}$$

We can now build the sparse matrix A. With $c = 1 + 4\,dt\,k$ and $m = -dt\,k$:

$$\mathbf{A} = \begin{pmatrix} \mathbf{D} & E & 0 & 0 & 0 & \cdots & 0 \\ E & \mathbf{D} & E & 0 & 0 & \cdots & 0 \\ 0 & E & \mathbf{D} & E & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & E & \mathbf{D} & E & 0 \\ 0 & \cdots & \cdots & 0 & E & \mathbf{D} & E \\ 0 & \cdots & \cdots & \cdots & 0 & E & \mathbf{D} \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} c & m & 0 & 0 & 0 & \cdots & 0 \\ m & c & m & 0 & 0 & \cdots & 0 \\ 0 & m & c & m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & m & c & m & 0 \\ 0 & \cdots & \cdots & 0 & m & c & m \\ 0 & \cdots & \cdots & \cdots & 0 & m & c \end{pmatrix} \quad \mathbf{E} = \begin{pmatrix} m & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & m & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & m & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & m & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 & m & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & m \end{pmatrix}$$

We solved this equation by using conjugate gradient solver from the Eigen library. We wrote matrix A as a sparse matrix object to benefit from the solver's optimization for sparse systems.

Project

After performing the three steps as in equation 1, the fluid might end up being non-divergence free. In order to solve this problem and to imply the fact that the fluid should be mass conserving at the end of every simulation we do a projection step. We use the Helmholtz-Hodge Decomposition (HHD) to decompose the field into the mass conserving and gradient fields. HHD states that any vector field can be decomposed into the sum of scalar field and divergence-free (mass conserving) vector fields. We apply the formula from the appendix B to get the divergence. We end up with another linear sparse system Ax = b, where x is the gradient field and b the divergence. We derive the coefficients of A as we did for Diffuse. We obtain c = –4 and m = 1. We use Eigen's solver again to get the gradient x. Finally, we subtract the gradient field from the previous velocity field to make mass conserving the new vector field, as explained in Appendix B.

Density equation

To solve the equation 2, we use the same first three steps as for the velocity. The difference between the two equations is the new term rate α * aS. We just replace the Project step by Dissipate where we divide the density by 1 + dt * aS, with aS being the dissipation rate.

Section 3. Results discussion

Running time

In our implementation of fluid simulation there are 5 main functions, namely AddForce, Transport, Diffuse, Project and Dissipate. We would like to present the running time (in microseconds) of these functions for the grid of size 64 * 64 and of size 128*128. We would like to note that the diffusion and projection steps take the majority of the running time, since solving linear equations (even with sparse matrices) takes more time than other steps.

AddForce	98	AddForce	56
Transport	2234	Transport	1 229
Diffuse	268 135	Diffuse	127 089
Project	6 929 306	Dissipate	82
Velocity step	7 209 011	Density step	132 365

*Table 1. Velocity and density step functions for 64*64 grid*

AddForce	21	AddForce	11
Transport	1 192	Transport	471
Diffuse	62 355	Diffuse	30552
Project	843 910	Dissipate	24
Velocity step	914750	Density step	33984

*Table 2. Velocity and density step functions for 128*128 grid*

User Interaction

In our program, the user is able to introduce the densities and provide the velocities through mouse interaction. A square of size 10*10 is filled with densities equal to 1 around the point at which the user pressed mouse, and the velocities of those points are calculated to be in the direction from the pressing to releasing points of the mouse.

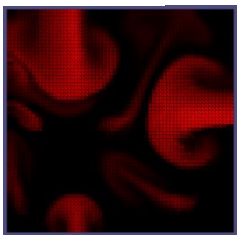


Figure 2. Densities introduced by user at different positions of the grid

Influence of parameters on behavior of fluid

When simulating the fluid, there are several parameters which can be changed and should have a huge effect on the behavior of fluid. In this section we show this influence by taking the parameters aS = 0.1 (dissipation rate), kS (diffusion constant) = 0.1, visc = 0.1 (viscosity), dt = 0.1 (time step) as default ones and we show the effect of each of them. All the screenshots below are taken after 30 iterations.

Time step

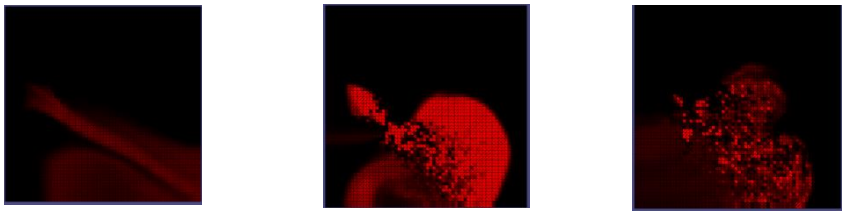


Figure 2. time step = 0.1, 0.5, 1.0

In our implementation of fluid simulation, we used the Runge-Kutta of second order, which was implemented in order to overcome any limitation of time step. However, by looking at figure above we can notice that our implementation fails with higher time steps.

Dissipation rate

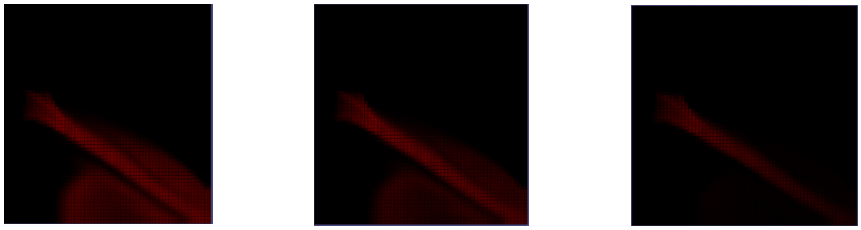


Figure3. dissipation rate = 0, 0.2, 1.0

Dissipation rate is used only for density step. At each dissipation step all the values of the density are divided by (1 + dt * dissipation rate). We can observe that by increasing the dissipation rate the density is reduced.

Diffusion constant

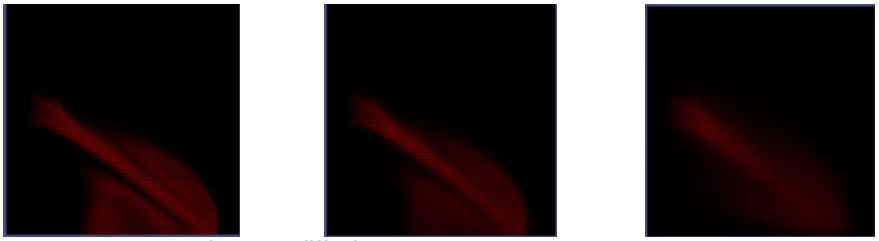


Figure 4. diffusion constant = 0, 0.2, 1.0

The diffusion constant shows how fast the fluid can diffuse in space. It can be observed that by increasing the diffusion constant the fluid spreads more in different directions, thus there are less cells of the grid with high density.

Viscosity

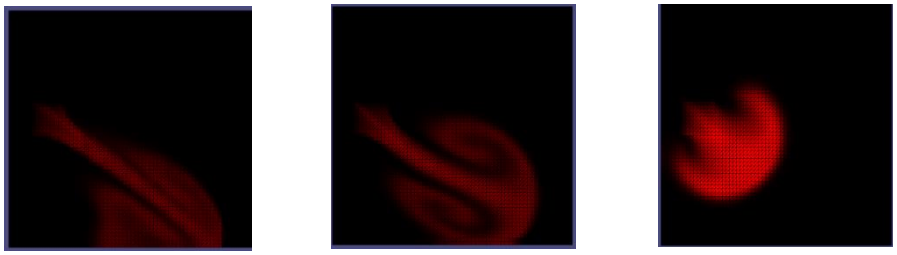


Figure 5. viscosity = 0, 10, 100

Value of viscosity shows the resistance of a fluid to deformation. By varying the viscosity of the fluid we can simulate various fluids. When the viscosity of the fluid is high, the fluid is hardly deformed. In contrast, fluids with low viscosity have low resistance and can be easily transformed by applying forces to it. In the figure 5 for viscosity = 0, we can see that the ideal fluid is easily transformed and follows the direction of the velocity provided by user. By increasing the viscosity, we can notice that it becomes more resistant to the applied forces.

In figures above the behaviour of the fluid on boundaries can also be observed. In our implementation, since the boundaries are fixed the fluid is reflected from the boundaries.

Section 4. Conclusion and possible extensions

We have implemented a fluid simulation which enables user interaction. The simulation is based on the resolution of the Navier Stokes equations. The external forces given by the user are first applied to the fluid. Then it is moved according to its velocity. It diffuses afterwards, and finally the density dissipates while the velocity is enforced to have zero divergence to keep the fluid mass-conserving. The method is stable even for high time steps thanks to a more robust backtracking. The simulation enables us to show the behavior of various fluids by changing the viscosity and diffusion rates.
The possible improvement of the method by Jos Stam could be saving values of density and velocity one and two time steps before the current time. This could allow us to improve the advection by changing the method of backtracking in time. Instead of using only using the values at previous time step, the values at two time steps before could be useful for more accurate calculations.