Fluid Properties

T Qi

December 17, 2023

1 Theory

Fluids are generalization of liquids and gases. To model and simulation their behavior, we will use the idea of fields.

Fields are structured data that models an n-dimensional space. Each field consists of a series of ordered grids/cells, with every cell being associated with a value: a scalar field has cells containing scalars, and a vector field has cells containing vectors. Grid values are denoted by indexing the field.

For a body of liquid, we associate both a velocity vector field and a substance density field. The velocity vector field denotes the macroscopic average molecule velocity within that grid space, and the density field denotes the average density of the substance within the grid. For a basic, non-compressible and non-viscous fluid simulation, we only need these two fields.

We notice the following three properties for fluids:

- Substances in fluid diffuse overtime, spreading until there is a uniform mixture across the entire liquid.
- The fluid contains velocity in its molecules that carries the substance along. This is called advection.
- The fluid can't be compressed, so liquids pushed against each other will be squeezed in the other dimension. This is the incompressibility assumption of fluids.

The above three properties, plus the direct user interference with the velocity and density fields, describes the iterative algorithm we will run to simulate fluids. That is, the field in the next time step is the sum of impacts from: the user, the diffusion, incompressibility, and advection.

Additionally, we wish for each of the steps to be local in state setting. This means that each grid can read arbitrary grid values, but should only set its own. This maintains the multithreading capabilities of the algorithm. The iterations should also be based on implicit modeling techniques rather than explicit solvers, for implicit solvers generally can handle larger time steps when compared to explicit solvers.

2 Field Interactions

The two important fields in solving for the incompressibility condition are the fluid velocity and pressure fields. We require the fluid velocity field to be divergence free at the end of each frame, as to maintain a field where substances are neither created or destroyed.

To maintain such a field V', we simply add the correction velocities V_P to the velocity field V each frame.

$$V' = V + V_P \tag{1}$$

The correction velocities are assumed to be caused by the pressure built up by the divergent velocity field. Notice that by our definition,

$$\nabla \cdot V' = 0$$

Firstly, we notice that a difference in pressure is sure to create a pressure-induced velocity from higher pressure cells to lower pressure cells. The relationship between the pressure scalar field to the pressure-induced velocity field is simply the gradient operator: it can be seen that the gradient of the pressure field is the negative pressure-induced velocity field

$$\nabla P = -V_P$$

This implies that, modifying equation **??**,

$$V' = V - \nabla P \tag{2}$$

Secondly, consider the divergence of the velocity field $\nabla \cdot V$. Intuitively, this is the net outflow of velocity for each cell (remember that positive divergence means a source and net outflow). Because velocities are caused by pressure difference, we assume that this is also the net pressure inflow into the cell. Hence, for the cell $v_{i,j}$ in velocity field V, the relationship between its divergence and the pressure field P is

$$(\nabla \cdot V)_{i,j} = (P_{i-1,j} + P_{i+1,j} + P_{i,j-1} + P_{i,j+1}) - 4P_{i,j} \tag{3}$$

With the pressure contribution from neighboring cells implicitly quartered to account for the lesser edge area.

To formally prove this, turn towards equation **??**, consider

$$\begin{aligned} V' &= V - \nabla P \\ V' + \nabla P &= V \\ \nabla \cdot (V' + \nabla P) &= \nabla \cdot V \\ \nabla \cdot \nabla P &= \nabla \cdot V \\ \Delta P &= \nabla \cdot V \end{aligned}$$

where we simplified the divergence of the new divergence-free velocity field on line 3, and used the Laplacian definition on the last line. The laplacian ΔP has the same discrete form as the intuitive pressure RHS above.

Importantly, we know the discretized version of the pressure laplacian and we can compute the velocity field divergences through first order approximations, meaning we can form a linear system of equations in solving for the pressure field.

2.1 Jacobi Iteration

To solve a specific type of linear system (specifically diagonal dominant systems), like the one in computing the pressure field, we can apply the Jacobi iteration method. In this algorithm, we iteratively solve for the pressure field cell values, with each iteration making a better estimate to the true pressure field values. Compared to the other Gauss-Seidel method, we store the new pressure field values in a buffer and only use the currently known field to compute the new pressures (unlike using the newly computed knowns in the GS method).

In general, for a system with unknowns x with the constraint $f(x)$ (vectorized unknowns and constraint functions), such that

$$x = f(x)$$

we apply the following iteration

$$x_0 = 0$$
$$x_{i+1} = f(x_i)$$

for as many times as we want. At the limit, the unknowns x_∞ should converge to

$$x_\infty = f(x_\infty)$$

In the context of our pressure field, we modify equation **??** to make the current cell at (i, j) the subject

$$P_{i,j} = \frac{1}{4}\left(P_{i-1,j} + P_{i+1,j} + P_{i,j-1} + P_{i,j+1} - (\nabla \cdot V)_{i,j}\right)$$

leading to a system like

$$P = f(P)$$

Applying the Jacobi method just needs us to maintain two buffers: one buffer for the current pressure, one buffer for the new pressure. We apply $f(P)$ and store the result in the new pressure buffer, and copy it to the existing buffer, then repeat. Over like 30 iterations, the present pressure buffer will approximate the true pressure field we wish.

We can also warm start the system by using the last pressure field estimate as P_0 for the new frame, for the pressure field shouldn't change too much between two immediate frames.

2.2 Projection

This then completes our algorithm for projection — removing divergence in the vector field by adding pressure induced velocities.

Using our estimates of the pressure field P, we can compute the pressure-induced velocity field by taking its gradient and negating, $-\nabla P$. Thus by equation **??**, we have the new velocity V' computed as

$$V' = V - \nabla P$$

3 Diffusion

Diffusion is the continuous spreading of substances in a fluid until a uniform density is achieved.

The explicit version of diffusion on a substance density field ρ is

$$\rho'_{i,j} = \rho_{i,j} + rh(\rho_{i-1,j} + \rho_{i+1,j} + \rho_{i,j-1} + \rho_{i,j+1} - 4\rho_{i,j})$$

where r is the rate of diffusion and h is the delta time.

To reformulate this implicitly, we let the next frame density ρ' be the field where its backwards diffusion is the current density field ρ. That is

$$\rho_{i,j} = \rho'_{i,j} - rh(\rho'_{i-1,j} + \rho'_{i+1,j} + \rho'_{i,j-1} + \rho'_{i,j+1} - 4\rho'_{i,j})$$

notice the minus sign in front of the diffusion equation: that is to simulate a backwards diffusion.

In rearranging for $\rho'_{i,j}$, we get

$$\rho'_{i,j} = \frac{1}{1+4rh}\left(\rho_{i,j} + rh(\rho'_{i-1,j} + \rho'_{i+1,j} + \rho'_{i,j-1} + \rho'_{i,j+1})\right)$$

the absence of any subtraction operation increases the stability of the simulation — for the density values cannot ever be negative if the initial conditions are positive. The higher stability does not imply higher accuracy, however.

This implicit equation can be solved the same way as the projection system. We can abstract this type of scalar field iteration with the following algorithm on each grid cell

$$V'_{i,j} = a(b(V_{i-1,j} + V_{i+1,j} + V_{i,j-1} + V_{i,j+1}) + cW_{i,j})$$

with V and W being scalar fields, and a, b, c are constants.

In the projection case

$$\begin{aligned} a &= \frac{1}{4} \\ b &= 1 \\ c &= -1 \\ V &= P \\ W &= \nabla \cdot V \end{aligned}$$

and in the diffusion case

$$\begin{aligned} a &= \frac{1}{1+4rh} \\ b &= rh \\ c &= 1 \\ V &= \rho' \\ W &= \rho \end{aligned}$$

In both cases, we are iterating to solve for V (which is P and ρ'). Everything else are constants.

4 Advection

We simulate advection in a gather operation by asking: what quantities have our current cell velocity carried from other cells through the frame.

The backwards advection equation is therefore

$$\rho'_{i,j} = \rho_{(i,j)-hV_{i,j}}$$

the intuition is that the substances at $(i,j) - hV_{i,j}$, will reach (i,j) in the frame.

In the case where $(i,j) - hV_{i,j}$ is not integer valued, we can (bi)linearly interpolate the nearby grid cells for an averaged density value.

Reminder to also advect the velocity field using the same algorithm:

$$V'_{i,j} = V_{(i,j)-hV_{i,j}}$$

5 Boundary Handling

We need to handle both static and dynamic boundaries in our simulation. They can be handled using similar methods.

Theoretically, we aim to let the boundary cells be controlled manually targeting a net zero outwards velocity due to pressure, no leaking of substance densities, and reflection in fluid velocities. We achieve this by first computing the surface normal of the boundaries, then to

- Set the boundary cell's pressure equal to the cell at its normal, as to create zero pressure induced velocities at the boundary.
- Set the substance densities at the boundary cells to be zero.
- Invert the velocity at the boundary cells against the velocity of the cell at its normal. This creates a reflective force on the fluid along the boundary.

We will simply apply these steps to all boundary (both static and dynamic) cells using a custom shader on each frame. We also let the normal fluid simulation shaders ignore such boundary cells.

In practice, start by storing and maintaining a cell type buffer that differentiates between fluid and wall type cells and another buffer for its corresponding barrier normals. For each frame, we follow the procedure to

1. Rebake the cell type buffer and its normals
2. Equalize the pressure field, zero the density field, and invert the velocity field at the current boundary cells
3. Normal fluid simulation on the non-boundary cells

To improve performance, we may store the types of boundary normals in a buffer and use a non-branching bit field to select the correct normal for each cell.

5.1 Dynamic Boundary

I currently have no idea how to implement the velocity contributions from to a moving dynamic boundary.

6 Conclusion

That's all for writing a semi-realistic fluid simulation. The aspects that could be improved in the future are

- Conservation of velocity and mass
- Dynamic boundaries
- Multigrid, or alternative implicit solvers
- 3D generalizations
- Web port