Animation & CGI Motion: Fluid Simulation

Theme 5 Milestone 1

Academic Honesty Policy

You are permitted and encouraged to discuss your work with other students. You may work out equations in writing on paper or a whiteboard. You are encouraged to use the wiki bulletin board to converse with other students, the TA, and the instructor.

HOWEVER, you may NOT share source code or hardcopies of source code. Refrain from activities or the sharing of materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this or previous years. We will be monitoring source code for individuality. Cheating will be dealt with severely. Cheaters will be punished. Source code should be yours and yours only. Do not cheat.

1 Introduction

In this theme, we will explore fluid simulation for animation. In particular, we will examine a method popular in the graphics community known as *stable fluids*.

The time evolution of a fluid is typically modeled with the *Navier-Stokes* equations. The *Navier-Stokes* equations are a collection of non-linear partial differential equations that, when solved, yield the velocity field that governs the motion of a fluid. The *Navier-Stokes* equations can be written in a number of forms – we are particularly interested in the viscous and incompressible case, as most everyday fluids of visual interest are viscous and incompressible. *Stable fluids* handles this particular case.

2 Implementation

Please read and implement the paper *Stable Fluids* by Jos Stam [3]. For further implementation details, please see Jos Stam's paper *Real-time fluid dynamics for games* [4]. *Stable Fluids* provides a nice description of the method, while *Real-time fluid dynamics for games* fleshes out a self-contained implementation. We suggest that you read *Stable Fluids* for a high-level overview of the method, and then mirror the implementation in *Real-time fluid dynamics for games*. The remainder of this text is intended to supplement the ideas presented in the reading, and assist with the implementation.

3 Usage

Click *right mouse button* and drag to draw densities on the fluids. Press *D* to switch between the normal and debugging view. See the example in Fig. 1.

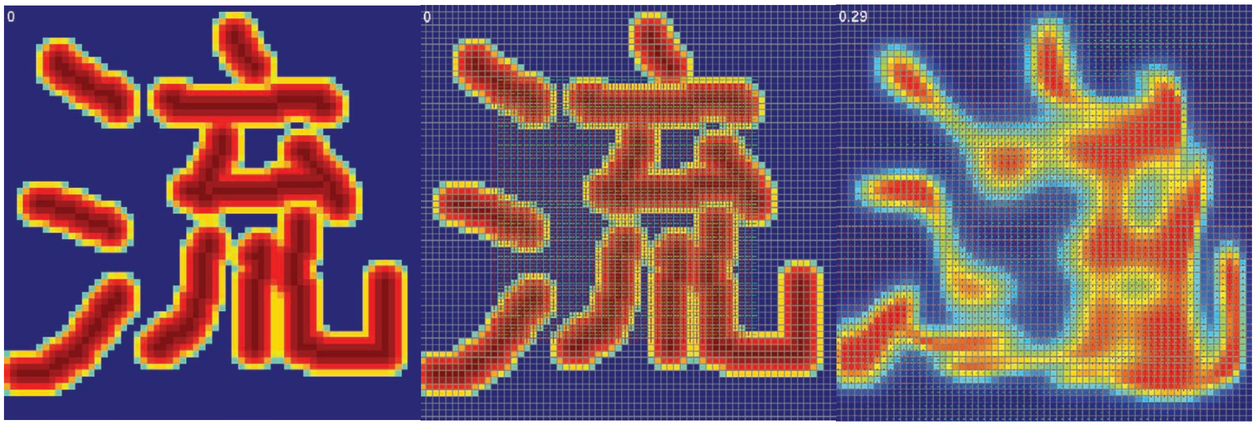

Figure 1: **Left:** use *right mouse button* to draw densities; **Middle:** press *D* to switch to debugging view; **Right:** velocities will be shown on each cell as line segments during simulation.

4 Staggered Grid (MAC Grid)

There are more than one style of grid we can use in an Eulerian simulation. In the the Stable Fluids paper, the grid we used was called a *collocated grid*, which means all the quantities such as velocities, pressure and dye density are defined at grid cell centers. This type of grid gives an intuitive velocity vector defined at cell centers, which points towards the fluid flow direction, and interpolations can be done easily. However when it comes to enforcing incompressibility, there is a serious disadvantage to it: discretizing the differential operator through finite differencing is a bit tricky. Central differencing gives a bad formula that only depends on neighboring values and not the value we are looking at, thus incorrectly returning zero for sawtooth a shaped field; forward/backward differencing is biased. See [1], section 2.4 Grids

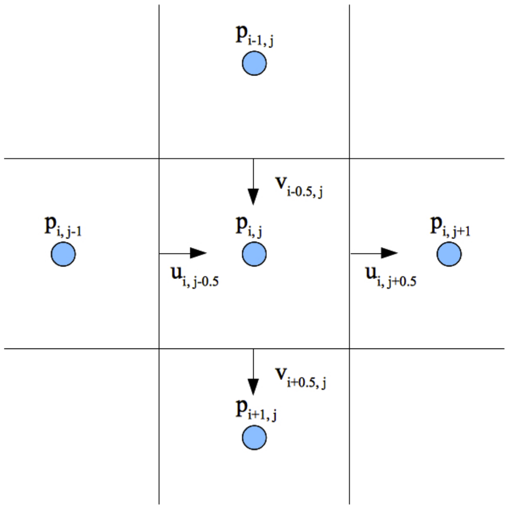

Figure 2: A staggered grid.

This motivates the *staggered grid*, first introduced in a 1965 paper by Harlow and Welch under the name

MAC grid (Marker-And-Cell), which is the approach we'll be using. The idea is to define pressure and dye density at cell centers, and velocities on cell walls. In our 2D case, it means the horizontal velocities are defined on vertical walls, and vice versa (shown in figure 2). With this grid, the differential operator is easily discretized:

$$\frac{\partial u_{i,j}}{\partial x} = \frac{u_{i,j+\frac{1}{2}} - u_{i,j-\frac{1}{2}}}{h} \tag{1}$$

where h is the width (and height) of a grid cell. Can you see why this is more elegant than what the collocated grid will give you? Here's a list of all the finite differencing discretizations we may need (assume the positive direction of u, x and j is pointing to the right, and positive direction of v, y and i is pointing down):

$$\begin{aligned}
\nabla \cdot \mathbf{u}_{i,j} &= \frac{\partial u_{i,j}}{\partial x} + \frac{\partial v_{i,j}}{\partial y} \\
\frac{\partial u_{i,j}}{\partial x} &= \frac{u_{i,j+\frac{1}{2}} - u_{i,j-\frac{1}{2}}}{h} \\
\frac{\partial v_{i,j}}{\partial y} &= \frac{v_{i+\frac{1}{2},j} - v_{i-\frac{1}{2},j}}{h} \\
\Delta p_{i,j} &= \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{h^2} \\
\frac{\partial p_{i,j+\frac{1}{2}}}{\partial x} &= \frac{p_{i,j+1} - p_{i,j}}{h} \\
\frac{\partial p_{i+\frac{1}{2},j}}{\partial y} &= \frac{p_{i+1,j} - p_{i,j}}{h}
\end{aligned}$$

The bold face $\mathbf{u}$ denotes the velocity vector (u, v), with the horizontal component being u and vertical component being v. Make sure you understand these formula well by looking at figure 2 and/or drawing your own diagrams. You'll need it when implementing this milestone.

One of the neat things about the staggered grid is, you can think of the velocities as flows through the walls it is defined on. Since the velocity is always perpendicular to the wall and defined the at the center of the wall, it naturally represents how much the fluid is flowing through that wall, into or out of the cell on either side. This view can be helpful when trying to understand incompressibility.

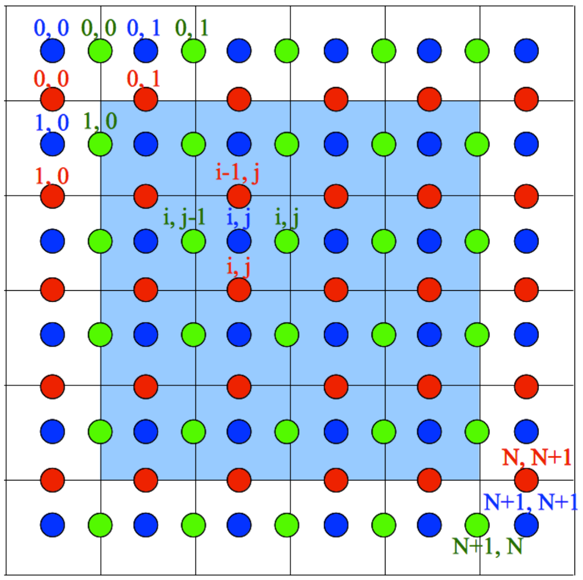

Figure 3: The indexing convention.

However if we want to convert the formula above to C++ code, a problem immediately shows up: we can't use fractions like $j + \frac{1}{2}$ as an array index. Of course we can choose to store it in the j-th element in the array, or the (j+1)-th element: it's pretty much an arbitrary decision. In the actual implementation, we have to define some convention for ourselves to follow, to map fractional indices to integer indices. We follow the convention outlined in figure 3. The blue dots are where we define pressure and dye density, having size (N+2) by (N+2). The green dots and red dots are horizontal (u) and vertical (v) velocities respectively, and have size (N+2) by (N+1) and (N+1) by (N+2) respectively. The indexing all starts from 0. As it can be seen clearly from the diagram, $u_{i,j+\frac{1}{2}}$ has index (i, j), and $v_{i+\frac{1}{2},j}$ has index (i, j) too, so equation 1 can be implemented as:

$$\frac{\partial u_{i,j}}{\partial x} = (u(i,j) - u(i,j-1))/h$$

The blue region in the grid represents fluid. We still have one extra layer of cells outside the actual fluid, acting as the solid boundary condition (discussed in the next section). Note that the red grid (vertical velocity v) does not include the bottom walls of the bottom row of solid cells (and top walls of the top row either), because we will apply the vertical boundary condition at the interface between the fluid cells and solid cells, and any vertical velocity beyond that is unnecessary. Same goes for the horizontal velocity grid.

Although the staggered grid shines at finite differencing, it does encounter some trouble when we want to sample the velocity at an arbitrary location. As explained in the Stable Fluids paper, this ability to sample velocity is needed the semi-Lagrangian method for advection: we trace a imaginary particle backward in time along the velocity field to its position at the previous time step, and read off quantities (density or velocity) there in order to fill them in the particle's current location, which requires interpolation of quantities at nearby grid points. Interpolation with staggered grid is a little bit trickier since grid points for different quantities are located at different positions in space, requiring a different interpolation routine for each quantity. The good news is, if we only look at dots of a single color and ignore other dots in figure 3, it is obvious that these dots still form a regular square grid, which means the bilinear interpolation still works, and we just need to offset the positions accordingly. Figure 4 demonstrates this idea. If you are interested in vertical velocities (defined at red dots) at the location of the yellow dot, you'll bilinearly interpolate the four red dots as shown on the right (see Algorithm 1 for example of interpolating U). Other quantities are similarly interpolated.

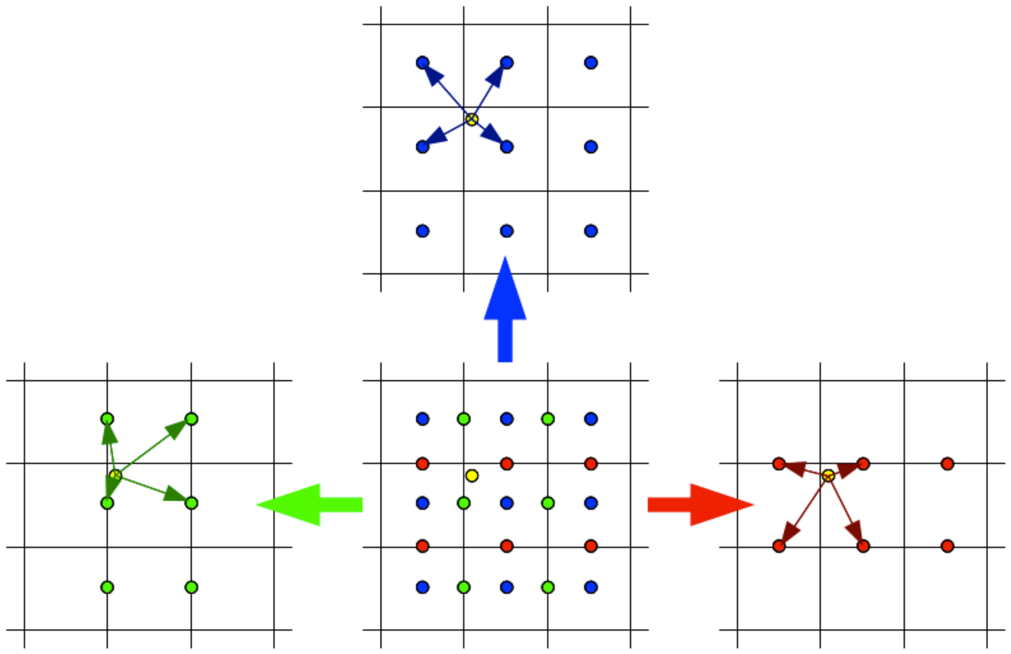

Figure 4: Interpolation for different quantities. We'd like to interpolate the red (v), green (u) and blue (density) values at the yellow dot's location. The grid nodes to use for interpolating each quantity is shown in a separate diagram.

Algorithm 1 Example of velocity interpolation

```
1:  procedure LERP($a, b, x$) return $(1 - x)a + xb$
2:  procedure INTERPOLATEU($u, i, j$)
3:      $i_1 \leftarrow \text{CLAMP}(integer(i)) \in [0, N]$
4:      $i_2 \leftarrow i_1 + 1$
5:      $j_1 \leftarrow \text{CLAMP}(integer(j - 0.5)) \in [0, N - 1]$
6:      $j_2 \leftarrow j_2 + 1$
7:      $s \leftarrow \text{CLAMP}(i - i_1) \in [0, 1]$
8:      $t \leftarrow \text{CLAMP}(j - j_1 - 0.5) \in [0, 1]$
9:      $u_1 \leftarrow \text{LERP}(u_{i_1,j_1}, u_{i_2,j_1}, s)$
10:     $u_2 \leftarrow \text{LERP}(u_{i_1,j_2}, u_{i_2,j_2}, s)$
11:     return LERP($u_1, u_2, t$)
```

5 Boundary Conditions

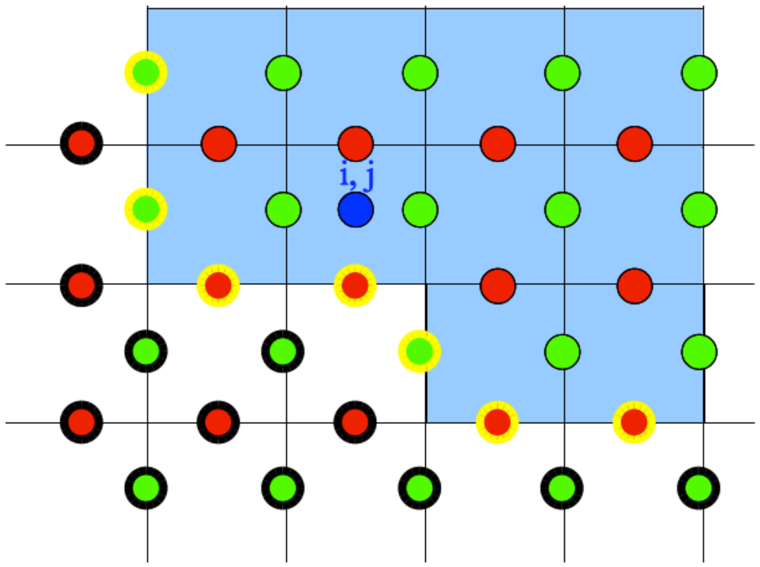

Figure 5: Solid boundary. Fluid cells are blue. Solid cells are white. The blue dot sits in the center of fluid cell (i, j). The velocities with thick yellow or black circles should be set to zero, and pressure gradient across walls with yellow circles should also equal zero.

When simulating fluids one must pay careful attention to boundary conditions. There are natural boundaries to the liquids we see every day, such as a solid container which doesn't allow to the liquid to pass and restricts its shape (*solid boundary condition*). As a simplification, instead of considering arbitrarily curved solid wall shapes and free surfaces, we assume that the solid boundaries we shall treat will align perfectly along the grid cells, and the same shall be true for a free surface. Therefore, any cell in the grid is always either a completely fluid cell, or a completely solid cell. There is no partially full state, and state switching is instantaneous. Furthermore, we assume solids to be static, so solid cells will remain solid throughout the entirety of the simulation. The goal is then as follows: given the current solid/fluid states of all the grid cells, advect the fluid so that it respects the boundary conditions (solid) and produces realistic simulations.

The condition we need to enforce at a solid wall is straightforward: no fluid flows into or out of a solid boundary. Generally, in the continuous setting, this is described by the following constraint:

$$\mathbf{u}_{fluid} \cdot \mathbf{n} = \mathbf{u}_{solid} \cdot \mathbf{n}$$

where $\mathbf{n}$ is the normal of the solid boundary. Normal relative velocity should be zero, while tangential relative velocity is allowed to be anything. Given our assumption that solids don't move, and solid boundaries are axis-aligned (either horizontal or vertical), this condition is simplified to

$$u_{fluid} = 0 \ for \ vertical \ walls,$$
$$v_{fluid} = 0 \ for \ horizontal \ walls$$

Therefore, when we project flow inside of our solver, we must ensure to do the following:

1. set the velocities that live normal to a solid cell to zero (the red and green dots with yellow borders in figure 5),

2. set the velocities that are inside solids to zero (the red/green dots with black borders in figure 5),
3. compute the divergence of the velocity field,
4. solve for the pressure from the divergence,
5. apply the pressure gradient to correct the velocities, only without overwriting those velocities above that have been fixed to zero.

Handling boundaries means we must also slightly modify the pressure solve. The pressure solve involves solving a poisson equation:

$$\Delta p = \nabla \cdot \mathbf{u}$$

Discretizing this for cell (i, j) (the blue dot's cell in figure 5), we get

$$\frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{h^2} = \nabla \cdot \mathbf{u}$$

Here is where our problem presents itself: what does $p_{i+1,j}$ mean, since it is inside a solid cell? This is a fictional pressure value resulting from our discretization, called a *ghost value*. Since it doesn't have any physical meaning, we can set its value to whatever that helps satisfy the boundary conditions - as if it's just an intermediate variable in the computation. Recall from Stable Fluids that after the pressure solve we use the pressure to correct the velocity field, namely by removing the divergent component:

$$\begin{aligned} v^{n+1}_{i+\frac{1}{2},j} &= v^{n}_{i+\frac{1}{2},j} - \frac{\partial p_{i+\frac{1}{2},j}}{\partial x} \\ &= v^{n}_{i+\frac{1}{2},j} - \frac{p_{i+1,j} - p_{i,j}}{h} \end{aligned}$$

Since $v^{n+1} = v^n = 0$ as a result of the solid boundary condition, $p_{i+1,j}$ can be determined from $p_{i,j}$, thus eliminating this degree of freedom entirely in the pressure solve. In practice, we can simply substitute this into the equation above and eliminate $p_{i+1,j}$:

$$\frac{p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 3p_{i,j}}{h^2} = \nabla \cdot \mathbf{u}$$

This same modified handling should appear in the diffusion step. As for the remaining step in the solver, advection, there is no need to change anything with regards to solid boundary conditions.

6 Starter Code

Please implement the various functions in FOSSSim/StableFluids according to the comments. We've created function placeholders according to the steps described in the Stable Fluids paper: advection, diffusion, and projection. Note that because we use the staggered grid instead of the collocated grid, advection and diffusion are implemented separately dependent on where the quantity being operated on is defined: the functions with suffix D operate on density (defined at cell centers), while those with suffix U and V operate on the horizontal and vertical components of the velocity (defined at vertical and horizontal wall centers) respectively.

7 User-Interface Controls

This milestone provides two execution modes. You can run the interactive mode and apply forces to velocity field, or you can load a predefined xml scene (option -s testfile.xml) that applies a prescribed velocity field. In the interactive mode, right-clicking and dragging deposits 'marker' fluid in the simulation grid. Left-clicking and dragging applies a force to the velocity field. The plus and minus keys switch between colormaps for visualizing the flow. The r key resets the simulation. If you decide to use one of the provided xml asset files, the left-clicking will be disabled but the rest of the controls remain the same.

8 Some Hints

1. Iterating 20 times in the Gauss-Seidel solver produces sufficient accuracy.

2. You may *CLAMP* (a procedure given in the starter code) the coordinates ($[(0,0),(N,N)]$ for density, $[(0,0),(N,N-1)]$ for u, and $[(0,0),(N-1,N)]$ for v) during interpolation. When dealing with the velocity, please be careful about the coordinates of a half cell (move by 0.5 in either horizontal or vertical axis) during the clamping.

3. Be careful with the offsets d applied to the coordinates during the advection process ($d = (0.5, 0)$ for u, $d = (0, 0.5)$ for v). For example, the following equation can be used for the advection of u:

$$\begin{aligned} b_i &= i + d_y - \Delta t N g_v(v, i + d_y, j + d_x) \\ b_j &= j + d_x - \Delta t N g_u(u, i + d_y, j + d_x) \\ u(i,j) &= g_u(u, b_i, b_j) \end{aligned} \tag{2}$$

   where g_u and g_v are the function for velocity interpolation about u and v.

4. The velocity on the solid boundary should be set to zero ($[(0,0),(N+1,0)]$, $[(0,N),(N+1,N)]$, $[(0,0),(0,N)]$, and $[(N+1,0),(N+1,N)]$ for u; $[(0,0),(0,N+1)]$, $[(N,0),(N,N+1)]$, $[(0,0),(N,0)]$, and $[(0,N+1),(N,N+1)]$ for v)

5. Although we are using a $(N+2)\times(N+2)$ grid, the cell size $h = 1/N$.

9 Further Reading

If you are interested in learning more about fluid simulation for animation, we recommend the SIGGRAPH 2007 Fluid Simulation Course Notes (and the book [1]) by Robert Bridson and Matthias Muller-Fischer: `http://www.cs.ubc.ca/~rbridson/fluidsimulation/`, as well as the PhD thesis of Jiang [2] for the discussion in more state-of-the-art techniques: `http://web.cs.ucla.edu/~cffjiang/thesis/thesis.html`.

10 Creative Scene

There is no support for creative scenes to be peer reviewed in this milestone.

References

[1] Robert Bridson. *Fluid simulation for computer graphics*. CRC Press, 2015.

[2] Chenfanfu Jiang. *The Material Point Method for the Physics-Based Simulation of Solids and Fluids*. PhD thesis, UNIVERSITY OF CALIFORNIA Los Angeles, 2015.

[3] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[4] Jos Stam. Real-time fluid dynamics for games. In *Proceedings of the game developer conference*, volume 18, page 25, 2003.