The most important milestone for fluid simulation is creating a **stable, divergence free velocity field**. Once the velocity field is stable, it is simple to advect arbitrary quantities within the grid, such as smoke density, heat, color, or whatever. By divergence free, we mean

• the volume of each cell remains constant each time step

• the velocity entering each cell should equal the velocity leaving each cell

• if we sum the rates of change in velocity in every direction, they should sum to zero, e.g.

∇⋅u=dudt+dvdt+dwdt=0∇⋅u=dudt+dvdt+dwdt=0

**MAC Grid**

Our first step is to understand the MAC Grid structure. The MAC Grid is a data structure for discretizing the space of our fluid. Quantities in the world, such as velocities and smoke density, will be stored at cell faces and centers. We use a staggered MAC Grid in particular because it allows us to estimate the derivatives (e.g. the rates of change in each direction) with more numerical stability (read the Bridson/Muller-Fischer notes to understand why!).

To start with something simple, create a 2x2x1 MAC grid as follows.

X faces are used to store the u-component of the velocity.

Y faces are used to store the v-component of the velocity.

Z faces are used to store the w-component of the velocity.

In our small MAC Grid above, we have 4 cells, 8 Z faces, 6 Y faces, and 6 X faces. We will store velocities at the faces and other quantities, such as smoke density, at the cell centers. Specifically, if we define the velocities with (u,v,w), we store the u values at X-faces, v values are Y-faces, and w values at Z-faces. The only tricky thing to remember is that *we will need to compute the full velocity (u,v,w) at each face location, but then only store one of the components there*. Implementation-wise, we have separate arrays, mU, mV, and mW for storing each velocity component. (i,j,k) indexing is used for each array of faces, for example,

• The front Z face indices are (1,1,0)(0,1,0)(1,0,0)(0,0,0). The back Z face indices are (1,1,1)(0,1,1)(1,0,1)(0,0,1).

• The Y face indices are (0,0,0),(0,1,0),(0,2,0) and (1,0,0)(1,1,0)(1,2,0).

• The X face indices are (0,0,0),(0,1,0) and (1,0,0),(1,1,0) and (2,0,0)(2,1,0).

In the code, indices in the Z direction correspond to *rows* and are indexed with *i*; indices in the Y direction correspond to *stacks* and are indexed with *j*; and indices in the X direction correspond to *columns* and are indexed with *i*. Next, we will compute a single advection and project step.

**Advection**

An empty MAC Grid, where all values are zero, corresponds to a grid with no velocity in it. Our first step is to add some movement to shake things up. To simplest way to start is to add a non-zero velocity to one of the internal faces of the grid. Let's add 1 unit of velocity in the Y direction, e.g. mV(0,1,0) = 1.

• What is the velocity at the center of each cell? Either (0,0,0) or (0,0.5,0)

Now that we've added some source velocity to the grid, we perform an advection step. Again, please refer to the notes if you're not sure of the details. Briefly, we go through each face in the grid and traceback to fetch the velocity that will be at our current location next step.

FOR\_EACH\_FACE

currentpt = position at center of current face

currentvel = velocity at center of current face

oldpt = currentpt - dt \* currentvel

newvel = velocity at old location

store one component of newvel depending on face type

Computing the velocities at arbitrary places in the grid is done by interpolating between the values stored at each face.

• What is the position of the Y-face? (0.5,1.0,0.5)

• What is oldpt? (0.5,1.0,0.5) - (0.1)\*(0,1,0) = (0.5, 0.9, 0.5)

• What is the velocity at position (0.5,0.9,0.5)? (0,0.9,0)

• What is the new value of mV(0,1,0)? 0.9

**Notes:**

• Because of traceback and interpolation, we never get a velocity that is larger than the values stored on the grid. This results in dissapation and loss of detail, but ensures that the system stays stable. Detail and dissapation can be counteracted by adding additional forces such as vorticity confinement.

• Parameter tweaking tip: make sure your timestep and velocities are not too large. In particular, you never want your traceback step to be larger than your grid size.

• What should you do when your traceback step takes you outside of the grid? If the position corresponds to a fluid source, just return the source value. You can also extrapolate the velocity from the nearest boundary, or set the value to zero (although this results in greater dissappation near boundaries)

**Projection**

After advecting the velocity, we can add external forces -- such as gravity, buoyancy, and vorticity confinement -- using a straight-forward euler step. Now, we have a new velocity field, but there's only one problem! It's not divergence free so anything we try to do to it will blow up. In this step, we will compute pressures so that when we apply them to the field the divergence-free property holds.

We will compute pressure values at the center of each cell which ensure that the resulting velocity field is divergence free.

The SIGGRAPH fluid course notes give a good description of the derivation of the Poisson equations that we will use to solve for pressure. To summarize, the values of our fluid at the next time step can be described in terms of our current (divergent, unstable) velocity field and the pressure,

un+1=u∗−δt1ρ∇pun+1=u∗−δt1ρ∇p

where u∗u∗ is our current velocity field, ρρ is the fluid pressure and is a parameter set by the user. In the above equation, the pressures are unknown. What should they be? We want the pressures that satisfy

∇⋅u=0∇⋅u=0

In other words, we want the rates of change in each direction of the cell to sum to zero, e.g. the amount of velocity entering the cell equals the amount leaving, e.g.

∇⋅u=dudt+dvdt+dwdt=0∇⋅u=dudt+dvdt+dwdt=0

where each of the derivatives above can be easily computed using finite differences between the faces on our grid. For example, to compute du/dt, we subtract the values stored at the X faces on each side of the cell and then divide by the cell size.

dudt=ui+12,j,k−ui−12,j,kΔxdudt=ui+12,j,k−ui−12,j,kΔx

dvdt=vi,j+12,k−vi,j−12,kΔydvdt=vi,j+12,k−vi,j−12,kΔy

dwdt=wi,j,k+12−wi,j,k−12Δzdwdt=wi,j,k+12−wi,j,k−12Δz

We will assume that our grid cells are square (e.g. Δx=Δy=Δz Δx=Δy=Δz). Now, we can write our divergence equation in terms of our current velocity field. For example, for a fluid cell surrounded by fluid cells, we would get

1Δx[ui+12,j,k−Δtρ(pi+1,j,k−pi,j,kΔx)−(ui−12,j,k−Δtρ(pi,j,k−pi−1,j,kΔx))+vi,j+12,k−Δtρ(pi,j+1,k−pi,j,kΔx)−(vi,j−12,k−Δtρ(pi,j,k−pi,j−1,kΔx))+wi,j,k+12−Δtρ(pi,j,k+1−pi,j,kΔx)−(ui,j,k−12−Δtρ(pi,j,k−pi,j,k−1Δx))]=01Δx[ui+12,j,k−Δtρ(pi+1,j,k−pi,j,kΔx)−(ui−12,j,k−Δtρ(pi,j,k−pi−1,j,kΔx))+vi,j+12,k−Δtρ(pi,j+1,k−pi,j,kΔx)−(vi,j−12,k−Δtρ(pi,j,k−pi,j−1,kΔx))+wi,j,k+12−Δtρ(pi,j,k+1−pi,j,kΔx)−(ui,j,k−12−Δtρ(pi,j,k−pi,j,k−1Δx))]=0

If we rearrange terms and put all our unknowns on the RHS and knowns on the LHS, we get the following for fluid cells that are surrounded by fluid neighbors. The RHS is the divergence of cell (i,j,k) and is computed with finite differences (just like above!).

(6pi,j,k−pi+1,j,k−pi−1,j,k−pi,j+1,k−pi,j−1,k−pi,j,k+1−pi,j,k−1)=−Δx2Δtρ(∇⋅u)(6pi,j,k−pi+1,j,k−pi−1,j,k−pi,j+1,k−pi,j−1,k−pi,j,k+1−pi,j,k−1)=−Δx2Δtρ(∇⋅u)

It's possible to derive similar expressions for fluid cells next to a boundary or solid. In general, the rules are

• The coefficient for pi,j,kpi,j,k is the number of fluid neighbors

• The coefficient for fluid neighbors is -1

• The coefficient for non neighbors is 0

Finally, we can derive an equation for every fluid cell in the grid. These can be combined into a large system of equations

Ap=bAp=b

Each of the rows of A represents the equation for a fluid cell. If we have 4 cells, A is a 4x4 matrix. The 4x1 column vector p is the pressure for each cell (our unknowns). The 1x4 column vector b is a function of the current divergence in each cell. Going back to our concrete example, our system of equations would look like

⎡⎣⎢⎢⎢2−1−10−120−1−102−10−1−12⎤⎦⎥⎥⎥⎡⎣⎢⎢⎢p1p2p3p3⎤⎦⎥⎥⎥=−Δx2Δtρ⎡⎣⎢⎢⎢∇⋅u1∇⋅u2∇⋅u3∇⋅u4⎤⎦⎥⎥⎥[2−1−10−120−1−102−10−1−12][p1p2p3p3]=−Δx2Δtρ[∇⋅u1∇⋅u2∇⋅u3∇⋅u4]

In this example, suppose Δx=1,Δt=0.1,ρ=1Δx=1,Δt=0.1,ρ=1. In this case, b is

b=−(1.0)2(0.1)∗(1.0)⎡⎣⎢⎢⎢⎢((0.9−0)+0+0)/1.00.0((0−0.9)+0+0)/1.00.0⎤⎦⎥⎥⎥⎥=⎡⎣⎢⎢⎢−9090⎤⎦⎥⎥⎥b=−(1.0)2(0.1)∗(1.0)[((0.9−0)+0+0)/1.00.0((0−0.9)+0+0)/1.00.0]=[−9090]

Now, we can plug in this equation into a linear solver such as matlab, to find values for p. It turns out that we get

p=⎡⎣⎢⎢⎢−3.375−1.1253.3751.125⎤⎦⎥⎥⎥p=[−3.375−1.1253.3751.125]

Now that we have values for p, we can compute our new velocity field.

u1,0,0=u∗1,0,0−ΔtΔxρ(−1.125+3.375)=−0.225u1,1,0=u∗1,1,0−ΔtΔxρ(1.125−3.375)=0.225v0,1,0=u∗0,1,0−ΔtΔxρ(3.375+3.375)=0.225v1,1,0=u∗1,1,0−ΔtΔxρ(1.125+1.125)=−0.225u1,0,0=u1,0,0∗−ΔtΔxρ(−1.125+3.375)=−0.225u1,1,0=u1,1,0∗−ΔtΔxρ(1.125−3.375)=0.225v0,1,0=u0,1,0∗−ΔtΔxρ(3.375+3.375)=0.225v1,1,0=u1,1,0∗−ΔtΔxρ(1.125+1.125)=−0.225

Note that from our advection step, u∗0,1,0=0.9u0,1,0∗=0.9 but all other velocities are 0. Also, we only compute new velocities for our interior faces because the boundary faces correspond to stationary objects and thus have velocity 0. The resulting velocity field swirls around like a pinwheel in the interior of the grid. The contents of mU = {0,-0.225,0,0,0.225,0}, mV = {0,0,0.225,-0.225,0,0}, and mW = {0,0,0,0,0,0,0}.   
**Notes:**

• Check that the velocity field after this step is divergence free by computing the divergence at each cell.

• If you use a uniform grid size, Δx=Δy=Δz Δx=Δy=Δz

• What is the velocity normal to a solid border? usolidusolid

• The fluid has free movement tangent to the boundaries.

• The resulting velocity field should be look smooth, no sudden changes!

**Putting it all together**

Once we have a divergence free velocity field, we can advect other properties in the field. For smoke, the fluid we simulate is the air, within which we would advect the smoke density (**carefull!** the smoke density is *different* from the air fluid density which we use in the projection step!) or temperature for buoyancy forces. Putting it all together, a typical simulation step might look like

void SmokeSim::step()

{

double dt = 0.01;

// Step0: Gather user forces

mGrid.updateSources();

// Step1: Calculate new velocities

mGrid.advectVelocity(dt);

mGrid.addExternalForces(dt);

mGrid.project(dt);

// Step2: Calculate new temperature

mGrid.advectTemperature(dt);

// Step3: Calculate new density

mGrid.advectDensity(dt);

}