Real-time fluid simulation with Navier-Stokes and a staggered MAC-Grid

William Ennefors*
Lulea University of Technology

05/03/2018

Abstract

Simulating fluids graphically can be difficult partly because the problem is really difficult to discretize, especially if the one who attempts to implement it has little experience in differential equations or vector calculus. The other difficulty with simulating fluids is displaying it in real-time, it's often inefficient and slow. This paper will describe how to write a simple real-time fluid simulation software with little to no prior experience in computational fluid simulations.

1 Introduction

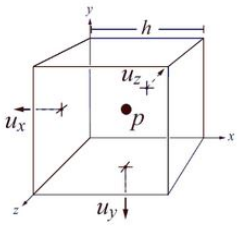
Fluids and liquids may at first glance seem like something impossible to understand, they seem to be chaotic, unruly and partly unpredictable. But when you start to think about it you can boil it down to a few terms and a velocity field. By utilizing the Navier-Stokes equation and a staggered MAC-Grid you can start describing how a fluid moves in space, this can make the problem a bit easier to understand and the actual implementation of the simulator starts to become more apparent and straightforward. All that is left is to actually understand how the Navier-Stokes equations and the MAC-Grid work in order to properly implement it. The MAC-Grid is rather simple, however, The Navier-Stokes equations are not that easy to understand if you have none to little experience in higher maths, the formulae can be confusing and you might not even understand some of the symbols and terms. There are a few concepts that can be difficult to grasp but in this paper, we will attempt; as someone who did not have prior knowledge in implementing a fluid simulation, to explain the different terms and the process of implementing it.

2 the MAC-Grid

First, before explaining the intricacies of Navier-Stokes, we need to explain what a staggered MAC-Grid actually is. The staggered MAC-Grid or Marker-and-Cell Grid is a way to help describe a fluid that is modeled as a velocity field or more explicitly a field of vectors. To simulate the fluid the velocity field is changed and evolves over time by moving marker-particles through space with the help of the field. Since velocities can't be stored at every single point in the field a grid with discrete points is used instead. This grid consists of cells with a width h; in this simulation, h is always at 1 unit in length, that can store the responding velocity $\mathbf{u} = (ux,\ uy,\ uz)$ values at the centre of its minimum faces; hence why it is a staggered grid and has a pressure variable stored in the middle. Our cells will also have a type and a layer to help with the algorithms.

*wilenn-5@student.ltu.se

Figure 1: A visual representation of a grid cell.



In order to make our simulation more efficient, we will build our grid so that it only surrounds the fluid and its immediate vicinity. This is done by storing the grid in a hash table with a hashing function proposed by [Worley 1996].

$$hash = 541x + 79y + 31z \tag{1}$$

With this hash table, we can easily add new cells and find and update/remove cells that are not needed we will also keep temporary vectors in each cell to store results which can't be done in place so we don't have to keep 2 synchronized grids updated at the same time.

3 Navier-Stokes

3.1 Navier-Stokes Equation Terms

The Navier-Stokes equations are two differential equations describe the velocity field of fluid substances over time.

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

What equation 2 says is that at any time, the amount of fluid flowing into any volume must be equal to the amount of fluid flowing out.

$$\frac{\delta \mathbf{u}}{\delta t} = -(\nabla \cdot \mathbf{u})\mathbf{u} - \frac{1}{\rho}\nabla p + v\nabla^2\mathbf{u} - \mathbf{F} \tag{3}$$

Equation 3 describes the motion of the fluid through space.
Now, this might seem to be scary when you have no experience in higher maths but when broken down into smaller more digestible parts it will become clearer.

$\frac{\delta \mathbf{u}}{\delta t}$ **The change of velocity over time.**

This is the derivative of velocity over time it will be calculated at all grid points containing fluid during each time step of the simulation.

$-(\nabla \cdot \mathbf{u})\mathbf{u}$ **The convection term.**

This term exists to conserve the momentum of the fluid, when the fluid moves through space the momentum also has to be moved together with the fluid, to do this a backward particle trace will be performed.

$-\frac{1}{\rho}\nabla p$ **The pressure term.**

This term describes the force generated from pressure differences. To get this force we will have to calculate the pressure gradient ρ is the density of the fluid cell, p is the pressure and ∇p is the pressure gradient. It bears to mention that in our simulation the density is set to 1.

$v\nabla^2\mathbf{u}$ **The viscousity term.**

This describes the thickness of the fluid, the higher value of the variable v the thicker the fluid is.
$\nabla^2\mathbf{u}$ will be calculated with a Laplace operator.

F **External Force.**

This term is any external force applied to the cell, Gravity is an example.

3.2 Operators

Now, to actually put these terms into code we need to explain the most important operators that will be used.

The gradient operator ∇. This operator will return a vector of partial derivates. The gradient is a vector field that tells how a scalar field changes in space. The pressure term $\nabla p(x, y, z)$ will need to have the backwards difference in the pressure field, where $\nabla p(x, y, z)$ is the pressure at the cell *(x, y ,z)* in the grid.

$$\begin{aligned}\nabla p(x,y,z) = (&p(x,y,z) - p(x-1,y,z),\\ &p(x,y,z) - p(x,y-1,z),\\ &p(x,y,z) - p(x,y,z-1))\end{aligned} \tag{4}$$

The divergence $\nabla\cdot$. We will use the divergence of a field to describe the net flow of fluid into or out of a cell in the grid. To do this we need to calculate the forward differences in the grid. When calculating the divergence it is important to note that *ux, uy* and *uz* is the individual velocity components of $\mathbf{u}$.

$$\begin{aligned}\nabla\cdot\mathbf{u}(x,y,z) = (&ux(x+1,y,z) - ux(x,y,z) +\\ &uy(x,y+1,z) - uy(x,y,z) +\\ &uz(x,y,z+1) - uz(x,y,z))\end{aligned} \tag{5}$$

The Laplacian operator ∇^2. The Laplacian operator is equal to the dot product of two gradient operator, this will evaluate a scalar value which describes the how much values in the original grid cell differs from its neighbors.

$$\begin{aligned}\nabla^2\mathbf{g}(x,y,z) = (&g(x+1,y,z) + g(x-1,y,z) +\\ &g(x,y+1,z) + g(x,y-1,z) +\\ &g(x,y,z+1) + g(x,y,z-1) -\\ &6g(x,y,z))\end{aligned} \tag{6}$$

There are a few important things to take note, however, when calculating the viscosity we must only use velocity components that border fluid cells this means that some terms will have to be dropped. Another thing to note is that to properly evaluate the difference of $\mathbf{u}(x,y,z)$ field we have to evaluate the Laplacian for u$x(x,y,z)$, u$y(x,y,z)$ and u$z(x,y,z)$ respectively.

$$\nabla^2\mathbf{u}(x,y,z) = (\nabla^2 ux(x,y,z), \nabla^2 uy(x,y,z), \nabla^2 uz(x,y,z)) \tag{7}$$

4 Simulation of the fluid

4.1 The first step, select the size of the time step.

To properly advance the fluid we need to select a specific time step, the time step needs to be small enough so the fluid doesn't seem chaotic and unnatural; this happens if the velocity of the fluid surpasses the width of a grid cell, but it also needs to be large enough so unnecessary computation will be made. This can be problematic and should not be underestimated. So in our simulation we chose this as a time step $\Delta t = u_{cfl} * \frac{h}{|\mathbf{u}_{max}|}$. This means, when advancing the field in real time the Δt that is used is the selected Δt and not the actual frame time so when rendering the fluid we will need to advance the fluid to the next displayed frame or else the simulation will seem slow and unnatural. The technique used to choose Δt is called the CFL condition and is a necessary condition for convergence while solving certain partial differential equations.

4.2 Updating the grid

Before calculating and advancing our fluid we need to update our dynamic grid. We'll do this by first updating the cells that contain fluid and then generate the buffer area of "Air" or "Solid" cells around the fluid so the fluid can actually move beyond the container cells. Algorithm 1 will describe this with pseudo code.

Algorithm 1 Grid generation and Update

```
set the layer of all existing cells to -1

Update cells that contain fluid
for each particle p do
    if cell C containing particle p exists then
        if C.type != solid then
            set layer of C to 0
            set type of C to fluid
    else
        if C is within grid bounds then
            create C and add it to the hashtable
            set layer of C to 0
            set type of C to fluid

for i = 1; i ≤ 2 do
    for each cell C do
        if C.type is fluid or air and C.layer = i - 1 then
            for each neighbour of C, N do
                if N exist in hash table then
                    if N.layer = -1 and N.type != solid then
                        set layer of C to 0
                        set type of C to fluid
                else
                    if C is within grid bounds then
                        create C and add it to the hashtable
                        set layer of C to i
                        set type of C to air
                    else
                        create C and add it to the hashtable
                        set layer of C to i
                        set type of C to solid

delete all cells which layer -1
```

4.3 Advancing the velocity field

All that is left now is to advance the velocity field. This is not an easy task, we will divide it into several sub-steps to make it more digestible.

4.3.1 Step 1, Apply convection

This is the first step, in order to convect we will need to perform a backward particle trace through the velocity field *i.e* move a hypothetical particle backward through the field for -Δt, and then replace the velocity of its previous location with the velocity of it's current. To get the velocity we'll need to interpolate the velocity of the particle separately for each velocity component *ux,uy* and *uz* this is done with RK2 (Runge-Kutta order 2) which is a half Euler-step.

$$\mathbf{y} = \mathbf{x} + \Delta t \mathbf{u}(x + \frac{\Delta t}{2}\mathbf{u}(\mathbf{x})) \tag{8}$$

To get, for example *ux*, the velocity needs to be interpolated from the position **Pos***(x,y-0.5,z-0.5)*. This cannot be done in place so this is where we'll make use of the temporary vectors in each cell to save the result and then copy over once all cells have been convected. This will also only be done for velocity components bordering fluid cells.

4.3.2 Step 2, Apply external forces.

After the field has been convected gravity vector $\Delta t\mathbf{g}(g_x, g_y, g_z)$ will be added to the velocity field. This will also only be applied to the velocity components bordering fluid cells.

4.3.3 Step 3, Apply viscousity.

To evaluate viscosity term we will make use of equation (5) to evaluate $\nabla^2\mathbf{u}$ then we'll add $\Delta tv\nabla^2\mathbf{u}$ to, as with both external force and convection, the velocity components bordering fluid cells this can also not be done in place and we'll have to save the result in the temporary vectors and then copy over the values once the calculations are done for each cell.

4.3.4 Step 4, Calculate and apply pressure

In order to properly solve the pressure so we fulfill the requirement set by $\nabla \cdot \mathbf{u}$ by solving the pressure properly, we will make it so that the divergence in the field is zero with our pressure term. However this cannot be done cell by cell, in order for the fields cells to have zero divergence the pressures have to be solved simultaneously, this will be done with a sparse linear system.

$$\mathbf{AP} = \mathbf{B} \tag{9}$$

Where $\mathbf{A}$ is a matrix containing coefficients, $\mathbf{P}$ is the vector with unknown pressures and $\mathbf{B}$ is a vector-based on the divergence of the field after step 3.

the method to fill $\mathbf{A}$ with coefficients is this: each fluid cell C_{ii} gets it's own row in the matrix, it's fluid neighbours get their corresponding coefficient on cell

Cs row set to -1 and the coefficient corresponding to C_{ii} is set to negative the number of non solid cells surrounding it.

B is also set in a similar way, cell C_i gets the position of i in the vector and the value is determined by the divergence of that cell *i.e*

$$b_i = \frac{\rho h}{\Delta t}(\nabla \cdot \mathbf{u}_i) - k_i p_{atm} \tag{10}$$

where k_i is the number of air cells surrounding cell C and p_{atm} is the atmospheric pressure, in this simulation it's set as 1. It is also important to note that the velocity going between the fluid and solid cells is considered to be zero. To solve this sparse system we used the math library Eigen and used its conjugate gradient solver but any good conjugate gradient solver may also be used.
to apply the solved pressures to our grid we'll update the velocities in the field with the help of the pressure gradient explained by equation 4.

$$\mathbf{u}_{new}(x,y,z) = \mathbf{u}(x,y,z) - \frac{\Delta t}{\rho h}\nabla p(x,y,z) \tag{11}$$

Where $\nabla p(x,y,z)$ is the pressure gradient. And as with viscosity and convection, the pressure is only applied to velocity components bordering fluid cells

4.3.5 Step 5, extrapolate the velocities

Now we that have the velocities of all our fluid cells we need to set the all unknown velocities in the rest of the grid. This is done by extrapolating the known velocities outwards. The procedure to do this is shown in algorithm 2.

4.3.6 Step 6, move the fluid particles

All that is left to do now is to move the particles through our velocity field. This is also done with RK2 interpolation to represent the fluid motion more accurately.

5 Results

The current simulation is stable and works as intended. Down below is a table with different settings on the fluid simulation and its performance in real time.

Algorithm 2 Grid Velocity Extrapolation

```
set the layer of all fluid cells to 0
set the layer of all non-fluid cells to -1

Update cells that contain fluid
for i = 1 to 2 do
    for each cell C that C.layer = -1 do
        if cell C has a neighbour N that N.layer = i-1 then
            for Velocity components of C not bordering fluid cells do
                set u_c to the average velocity of Cs
                neighbours that N.layer = i-1.
            C.layer = 1
```

nrOfCells	nrOfParticles	FPS
400	2500	60
400	10000	30
400	250000	5
800	10000	28

6 Related work

This simulation was made possible with a these papers as help. First, the main papers that this was based on [Foster and Fedkiw 2001] and [Cline, Cardon and Egbert 2005]. These are the most influential papers that helped tremendously during the actual implementation process and research phase. Most of the methods we have used are proposed by their work.

7 Conclusion

As this was the first time implementing something of this complexity we're rather satisfied with our result. The simulation looks realistic enough and runs at a decent frame rate. There are, however a few improvements that we would like to do at some later date. These would be to do calculations of the velocity field on the GPU to make the simulation run even better. We would also like to utilize the Pic/Flip method to improve the accuracy and completeness of the simulation and the last improvement would be to implement a surface tracking algorithm to make the

fluid seem like actual water instead of using simple particles to represent the fluid.

8 References

(1) WORLEY, S. 1996. A cellular texture basis function. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, ACM Press, 291294.

(2) FOSTER, N, AND FEDKIW, R. 2001. Practical animation of liquids. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques,, ACM Press., 2330.

(3) STEELE, K., CLINE, D., AND EGBERT, P. 2005. Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics

(4) BARBA, L. 2013. 12 steps to Navier-Stokes @ http://lorenabarba.com/blog/cfd-python-12-steps-to-navier-stokes/