UNIVERSIDAD PANAMERICANA

FLUID SIMULATION

COM 139: SIMULATION & VISUALIZATION [1]

1 INTRODUCTION

In the gaming industry, fluid flows are very common. From rising somoke, clouds, mists, rivers, oceans, etc. Given that one of the main goals of a game engine is to produce an immersive experience for the players, it is desirable to include fluid flows into the engine. There are already several tecniques that attempt to simulate a fake fluid-like effect, such as particles rendered as textured sprites; though achieving a realistic effect is not easy.
Some people suggested that a better alternative would be to use physics of fluid flows which have been developed since the mid 1700's. The Navier-Stokes Equations are a precise mathematical model for most fluid flows occurring in nature. These equations, however, only admit analytical solutions in simple cases. No progress was done further on this field until researchers were able to use computers to develop algorithms to solve these equations in the 1950's. These algorithms were designed to be physically accurate, since they were used in the design of real world elements that required this precision for safety reasons. But this required a lot of complex calculations and a big time investment.
In Computer Graphics and in games, what matters most is that it looks ralistic and that it is fast. Thus, we want to have a simple solver that can be implemented on a standard PC or console.

2 THE PHYSICS OF FLUIDS

Mathematically, the state of a fluid at a given instant of time is modeled as a velocity vector field: a function that assigns a velocity vector to every point in space. Imagine the air that occupies a room, its velocity will vary due to the presence of heat sources, air drafts, etc. For example, the velocity of the air near a radiator will predominantly be pointing in the upward direction due to heat rising. The distribution of velocities within a room is also quite complex as is evident when watching the smoke rising from a cigarette or the motion of dust particles in the air. The Navier-Stokes Equations are a precise description of the evolution of a velocity field over time. Given the current state of the velocity and a current set of forces, the equations tell us precisely how the velocity will change over an infinitesimal time step. Equation 1 (top) depicts these equations in a compact vector-like notation. Very roughly the equation states that the change in velocity is due to the three terms on the right hand side of the equal sign.

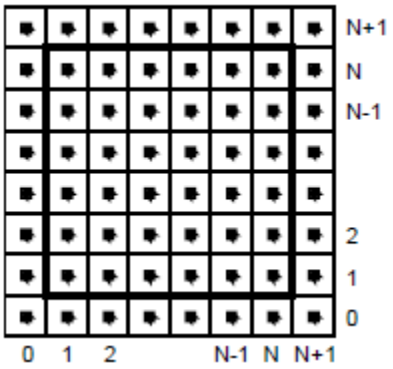
$$\begin{aligned} \frac{\partial u}{\partial t} &= -(u \cdot \nabla)u + \nu\nabla^2 u + f \\ \frac{\partial \rho}{\partial t} &= -(u \cdot \nabla)\rho + \kappa\nabla^2 \rho + S \end{aligned} \tag{1}$$

A velocity field on its own isn't really visually interesting until it starts moving objects such as smoke particles, dust or leaves. The motion of these objects is computed by converting the velocities surrounding the object into body forces. Light objects such as dust are usually just carried along with the velocity

[1] *Engeneering Faculty, Universidad Panamericana, Guadalajara, México*

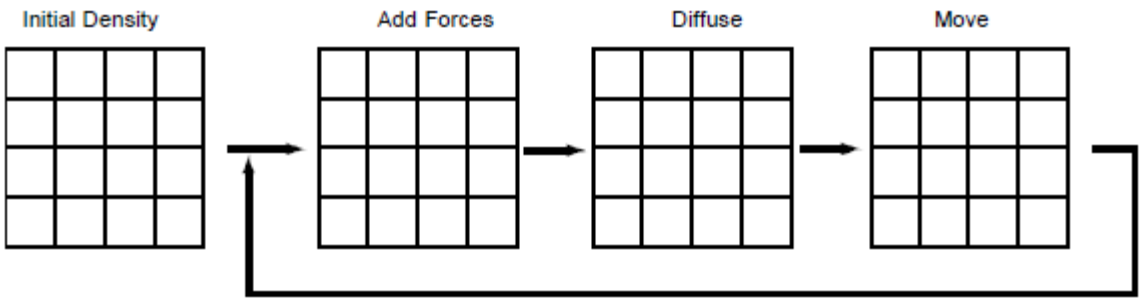
field: they simply follow the velocity. In the case of smoke, it is prohibitively expensive to model every particle. Hence in this case the smoke particles are replaced by a smoke density: a continuous function which for every point in space tells us the amount of dust particles present. The density usually takes values between zero and one: where there is no smoke the density is zero, and elsewhere it indicates the amount of particles present. The evolution of the density field through the velocity field of the fluid can also be described by a precise mathematical equation described at the bottom of eq 1. It should be evident to anyone that the two equations in Figure 1 look very much alike.

3 A FLUID IN A BOX

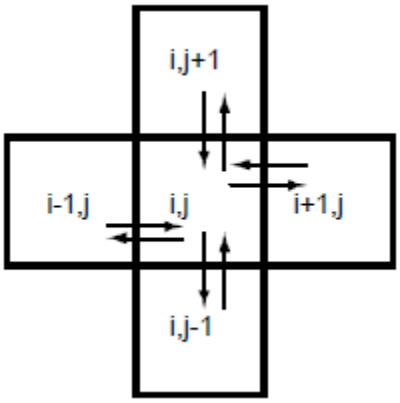


Mathematical equations for fluids are useful when thinking about fluids in general. However, in practice we need a finite representation for our fluids. The usual approach is to dice up a finite region of space into identical cells and sample the fluid at each cell's center. Therefore, our fluid will be modeled on a square grid like the one shown in the figure above. We allocate an additional layer of grid cells around the fluid's domain to simplify the treatment of the boundaries. Both the velocity and the density are assumed to be constant in each grid cell and we usually display their values at the cell center.
The basic structure of our solver is as follows. We start with some initial state for the velocity and the density and then update its values according to events happening in the environment. Forces will set the fluid into motion while sources will inject densities into the environment. In a game the forces could come from a virtual fan, a creature blowing air or a falling object, while the density sources could be located at the tip of a burning cigarette or at the top of a smoke stack. The simulation is therefore a set of snapshots of the velocity and density grids. We assume that the time spacing between the snapshots is given by the fixed variable dt.

4 MOVING DENSITIES

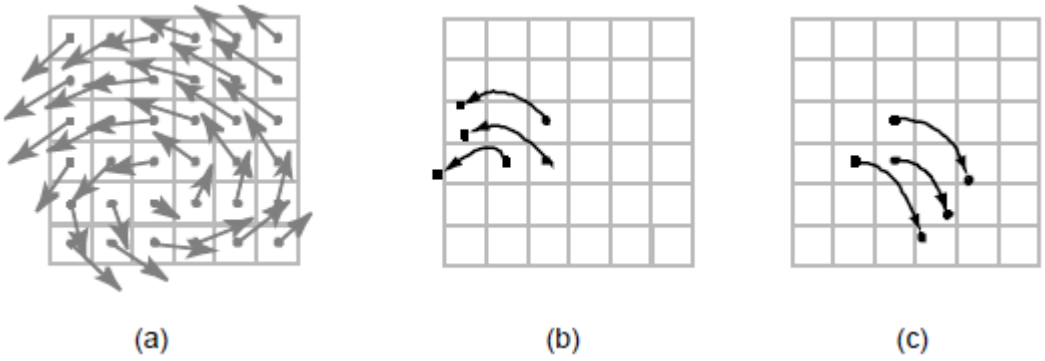


As explained above we will first describe the solver for a density field moving through a fixed velocity field that doesn't change over time. Let us consider the density equation again depicted in the bottom of Equation 1. This equation states that the changes in density over a single time step are due to three causes. These causes are the three terms on the right hand side of the equal sign in the equation. The first term says that the density should follow the velocity field, the second states that the density may diffuse at a certain rate and the third term says that the density increases due to sources. The solver will resolve these terms in the reverse order as they appear in the equation as shown in the figure above. We start from an initial density and then repeatedly resolve these three terms over each time step.
The first term is easy to implement. We assume that the sources for a given frame are provided in an array s[]. This array is filled in by some part of the game engine which detects sources of density.



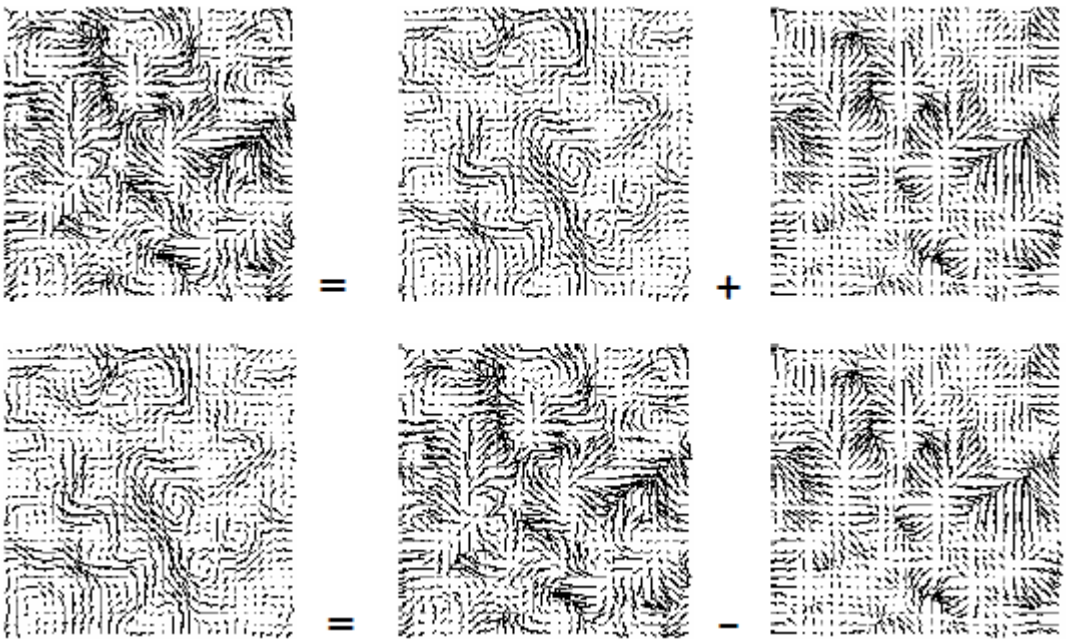
The second step accounts for possible diffusion at a rate $\texttt{diff}$, when $\texttt{diff} > 0$ the density will spread across the grid cells. We first consider what happens at a single grid cell. In this case we assume that the cell exchanges densities only with its four direct neighbors as shown above. The cell's density will decrease by losing density to its neighbors, but will also increase due to densities flowing in from the neighbors, which results in a net difference of $x_0[i-1,j] + x_0[i+1,j] + x_0[i,j-1] + x_0[i,j+1] - 4 * x_0[i,j]$.
We will also introduce the factor a that will represent the diffusion rate ν in our formula. a will be calculated as $\texttt{dt} * \texttt{diff} * N^2$. For large diffusion rates a the density values start to oscillate, become negative and finally diverge, making the simulation useless. This behavior is a general problem that plagues unstable methods. For these reasons we consider a stable method for the diffusion step. The basic idea behind our method is to find the densities which when diffused backward in time yield the densities we started with. We could build the matrix for this linear system and then call a standard matrix inversion routine. However, this is overkill for this problem because the matrix is very sparse: only very few of its elements are non-zero. Consequently we can use a simpler iterative technique to invert the matrix. The simplest iterative solver which works well in practice is Gauss-Seidel relaxation. The beauty of this version of the diffusion solver is that it is almost as simple as the unstable one, but can handle any values for $\texttt{diff}$, $\texttt{dt}$, or N: no matter how big these values are the simulation will not blow up.
Let us now turn to the final step in the density solver which forces the density to follow a given velocity field. Similarly to the diffusion step we could set up a linear system and solve it using Gauss-Seidel relaxation. However, the resulting linear equations would now depend on the velocity, making it trickier to solve. Fortunately, there is an alternative which is more effective. The key idea behind this new technique is that moving densities would be easy to solve if the density were modeled as a set of particles. In this case we would simply have to trace the particles though the velocity field. For example, we could pretend that each grid cell's center is a particle and trace it through the velocity field as shown in Figure (b). The problem is that we then have to convert these particles back to grid values. How to properly do that is not necessarily obvious. A better method is to find the particles which over a single time step end up exactly at the grid cell's centers as shown in Figure (c). The amount of density that these particles carry is simply obtained by linearly interpolating the density at their starting location from the four closest neighbors.

This suggests the following update procedure for the density. Start with two grids: one that contains the density values from the previous time step and one that will contain the new values. For each grid cell of the latter we trace the cell's center position backwards through the velocity field. We then linearly interpolate from the grid of previous density values and assign this value to the current grid cell.



5 EVOLVING VELOCITIES

We are now in a position to present the velocity solver. Once again consider the equations in Equation 1. In the light of what we now know about the density solver we can interpret the velocity equation as saying that the velocity over a time step changes due to three causes: the addition of forces, viscous diffusion and self-advection. Self-advection may seem obscure but we can simply interpret it as the fact that the velocity field is moved along itself. More importantly we can now reuse the routines that we developed for the density solver and apply them to update the velocity field.



The idea is to make it mass conserving in the last step. To achieve this we use a result from pure mathematics called the Hodge decomposition: every velocity field is the sum of a mass conserving field and a gradient field. This result is illustrated in the figure above (top). Notice how the mass conserving field has nice swirly-like vortices, typically the type of field we would like to have. On the other hand the

gradient field shown in the upper right corner of the figure is the worst possible case: the flow at some points either points all outward or inward. In fact the gradient field indicates the direction of steepest descent of some height function. Imagine a terrain with hills and valleys with an arrow at every point pointing in the direction of steepest descent. Computing the gradient is then equivalent to computing a height field. Once we have this height field we can subtract its gradient from our velocity field to get a mass conserving one as shown in the figure (bottom). We will not go into the hairy mathematical details, but will simply state that computing the height field involves the solution of some linear system called a Poisson equation. This system is sparse and we can re-use our Gauss-Seidel relaxation code developed for the density diffusion step to solve it. Here is the code for the projection step

6 THE TASK

Given that you are already provided with a fully working solution, you will have 4 tasks for this project. The goal of them is for you to understand in detail how the code works, where can you modify it and then implement the requested behvior.

1. Add the capability to the simulator to create multiple sources for velocity and density without modifying the source code. This means that you can either receive them via command line or via a file. Either is accepted, you DON'T have to do both.
2. Animate the velocity forces. This should also be configurable via a command line parameter or can be included in the input file. I would expect at least 2 behaviors.
3. Create color schemas for the simulation. Right now it is in yellow and blue, but I would want you to have multiple coloring options.
4. Simulate the presence of objects in the simulation. As before, this could be provided via command line parameter or via file.

Each of the points requested above have a 25% weight for the grade of this project.

6.1 What to deliver

- Source code
- Development log. This is just a document with the explanation of the challenges you faced and how you solved them. It is perfectly OK to state that some of the challenges were not solved. If you refer to other sources to solve a problem, please don't forget to cite them properly.
- At least 5 videos that showcase the ability of your solution using all the options above in the combination you want.