Fluid Advection Techniques

Miguel Ángel Pazos Crespo

July 18, 2012

Contents

Chapter 1

Introduction

1.1 Presentation

Fluid simuation is nowadays a very important tool in computer graphics. It is used to simulate water, smoke and other fluid phenomena in the film industry, as well as in shipmaking industry, medical simulation and increasingly in real-time software and video game industry. The techniques used in fluid simulation are different depending on the purpose, e.g. software used in engineering is designed to simulate experiments in order to retrieve very precise data, so data accuracy and interpretability are more important than visual realism. On the other hand, in film industry accuracy is not so important but the fluid must look real to the human eye, both physically and visually. This project focuses in the latter. Both classic and state of the art techniques will be evaluated and compared in order to give an overview of the current state of fluid simulation. Also, a modular fluid solver will be developed, providing a framework for the evaluation of different fluid simulation methods.

1.2 Overview

The techniques used in computation fluid dynamics are based mainly in two mathematical models. The fist one defines the fluid as a continuum whose state is described by the following macroscopic variables:

- Density ρ
- Velocity $\vec{u}$
- Pressure p
- Temperature T

- Energy E

A series of conservation equations, called Navier Stokes Equations, define the evolution of these fluid properties over space and time. There are many techniques intended to solve these equations, e.g. SPH, PIC/FLIP, Semi Lagrangian Advection and so on. The other model describes the fluid as a set of particles. It defines the way the particles move and collide using statistical models. Lattice Gas Automata and Lattice Boltzmann Method are based on this model. This project is focused on the first group. The Navier Stokes Equations will be explained and some of the most important techniques for solving these equations will be reviewed and compared. Also, a brief overview on the implementation of the modular solver will be given.

It is important to note that this project will focus on isothermal and incompressible fluids. That means that temperature and energy will not be taken into account and also that the simplified version of Navier Stokes Equation for incompressible flows will be used.

1.3 Related Work

One of the earliest and most important works in this field was the one published by Foster and Metaxas [FN96]. In their work they presented an Eulerian method using a grid-based discretization and finite differences to solve the Navier Stokes Equations. In 1999 Jos Stam published a very efficient and stable method for solving the advection part of the NS equations called Semi-Lagrangian Advection [Sta99]. Stam's method has an excessive numerical dissipation so there have been several other works trying to mitigate these problem, like Foster and Fedkiw did in 2001 using a higher order interpolation method and adding vorticity confinement [FN01]. They also introduced a level set method for surface tracking using particles to measure volume loss. In 2002, Enright et al. [DE02] improved the particle level set method, using particles on both sides of the fluid interface. In 2007, Selle et al. [A.07] developed an improved advection method making use of other existing methods like Stam's Semi Lagrangian Advection. Another important work was Zhu And Bridson 2005 [ZY05], were they recovered the PIC method of Harlow [H.86] and the FLIP method of Brackbill and Ruppel [BJU86] to present an advection method with very little numerical dissipation. More recent works have been focused on improving the efficiency of existing methods like Lentine,Zheng and Fedkiw who presented a method for solving the projection step in a coarse grid thereby reducing the cost of the most expensive step of the fluid simulation [LM10]. Other recent example is Chentanez and Müller [CN11] who modified the grid discretization in order to make it coarser in the deep zone of the fluid where less detail is required.

That was for Euluerian methods so far. On the other hand, Lagrangian methods had also been present in the fluid simulation scene, specially since Monaghan introduced the SPH method in 1992 [J.92]. Müller et al. [MM03] applied SPH to water simulation

in 2003 and Takeshita et al. [D.03] used a pure particle approach for fire simulation. About hybrids methods, in addition to the aforementioned PIC/FLIP, it is worth to mention the use of MPM (Material Point Method) for simulating deformable materials [SD95].

1.4 Goal

In this project, we will test and compare three different advection methods, namely, Semi-Lagrangian Advection, MacCormack Advection, and PIC/FLIP. To do that, we will develop a complete fluid solver from scratch in a modular fashion so that the different techniques can be developed independently and it is easy to switch between one technique and another.

We will start by developing a fluid solver using Semi-Lagrangian advection as originally described in [Sta99] but using a 2D staggered grid, as in [R.08], which will be the framework in which the rest of the techniques will be implemented. Once this is done, and using the newly developed SLA, we will implement MacCormack advection scheme [A.07]. After that, we will step into hybrid methods implementing the PIC/FLIP method described in [ZY05]. As we will see in 3.5, PIC/FLIP takes advantage of the fact that particles methods are better suited than grid methods in solving the advection step to create a hybrid method able to solve accurately the advection and non advection terms of the fluid equations. Once the three methods are implemented, the results obtained will be compared to each other so that we can understand the advantages and disadvantages of every method. The PIC/FLIP solver will be extended to 3D so, in the end we will have a hybrid fluid solver able to simulate 3D high quality fluids and a good base for latter development of new techniques.

Finally, we will review some techniques that could be used to improve other aspects of the solver in the future such as rendering, fluid surface representation or optimization.

Chapter 2

Physics of Fluids

2.1 Navier-Stokes Equation

The Navier-Stokes equations model the fluid motion by describing how the velocity field changes over time. Here is the incompressible version of these equations, i.e assuming volume and density constant:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{g} + \nu \nabla \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p \tag{2.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.2}$$

The first term of the left-hand side of the equation is called *material derivative*:

$$\frac{D \vec{u}}{Dt} = \frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \tag{2.3}$$

The material derivative of the velocity field is the sum of the rate of change of the velocity in an infinitesimal volume element of the fluid: $\frac{\partial \vec{u}}{\partial t}$ plus the rate of change of the velocity due to the movement of the fluid itself: $\vec{u} \cdot \nabla \vec{u}$. A more detailed explanation about the material derivative is given in Apendix A. Equation 2.1 can be interpreted as the Newton's second law applied to an incompressible fluid if it is arranged like this:

$$\frac{D \vec{u}}{Dt} = \vec{g} + \nu \nabla \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p \tag{2.4}$$

ice that mass has been taken out from both sides of the equation. The right-hand side of the equation is the sum of the *forces* acting on the infinitesimal element of fluid.
The first of these forces is gravity which influences all fluid elements with the same intensity and direction. After gravity comes viscosity which depends on the Laplacian of the velocity and on a parameter ν named *kinematic viscosity*. This parameter is an

indicator of the fluid's reluctance to deform under the action of external forces. The Laplacian operator roughly measures if there's a local maximum or minimum of velocity in the neighbourhood of a point, and the viscosity attenuates this local extremes thus smoothing the velocity field.
The last force is a force caused by the pressure exerted by the neighboring fluid elements. This force goes in the opposite direction to the pressure gradient, which means that higher pressure areas exert a force on lower pressure areas. The magnitude of this force depends on both the magnitude of the pressure and the density of the fluid.

Equation 2.2 is the incompressibility condition, which will be explained in 2.4.

2.2 Lagrangian and Eulerian representations

The equations describing the motion of a continuum like a fluid can be defined in two ways depending on the point of view: Eulerian or Lagrangian. As we are going to see in detail in the next section, the equations will be different depending on the point of view.

2.2.1 Lagrangian

In the Lagrangian point of view, the fluid is seen as a set of infinitesimal elements called particles. Every particle has a position, $\vec{x}$ and a velocity $\vec{u}$, and each particle moves following its own velocity. If we return to the definition of material derivative, the rate of change of the velocity due to the motion of the fluid itself is zero, since the particles are already changing their positions as the fluid moves. The material derivative would be therefore:

$$\frac{D\,\vec{u}}{Dt} = \frac{\partial\,\vec{u}}{\partial t} \tag{2.5}$$

2.2.2 Eulerian

In the Eulerian point of view a static grid of points is defined and the fluid properties are then calculated only for those fixed points. As these points do not move with the fluid, the rate of change of velocity will depend on both the rate of change of velocity with time and the rate of change due to the movement of the fluid itself along the static grid. The material derivative then would be as it was presented at the begining of the chapter:

$$\frac{D\,\vec{u}}{Dt} = \frac{\partial\,\vec{u}}{\partial t} + \vec{u}\cdot\nabla\,\vec{u} \tag{2.6}$$

Although the equations defined with the Lagrangian point of view look simpler, the solver developed in this project will stick mostly to the Eulerian point of view because

it is simpler to make a numerical aproximation of the spatial derivatives using a fixed grid and also because the boundary conditions are easier to implement accurately with this approach.

2.3 Viscosity

As we have explained before, Navier-Stokes equations stablish how viscosity affects the fluid motion. When designing computer generated fluid animations, we usually want to create visually atractive animations which show the small scale effects such as small swirlings and turbulences. Viscosity hides all these details, as it opposes the fluid motion.

As we will see later on, the numerical methods which we are going to use in order to solve the Navier-Stokes equations create some numerical viscosity, so we can leave aside the viscosity term in the Navier-Stokes equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{g} - \frac{1}{\rho}\nabla p \tag{2.7}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.8}$$

These simplificated equations are known as *Euler Equations*.

2.4 Incompresibility

Equation 2.2 stablishes the incompressiblity condition of the fluid. The divergence operator tells us how much the fluid is concentrating in a single region or flowing out of it, in fact, as it will be shown in B, the divergence of the velocity is approximately the gain of fluid per unit volume in an infinitesimal fluid element. If the fluid is perfectly incompressible, the volume should be always constant and therefore it should not be concentrating or flowing out of any point, which would mean that the volume is expanding or contracting. For that reason the incompressibility condition states that the divergence of the velocity vector field must be zero.
If we were modelling a compressible fluid, these condition would not exist and it should be necessary to add an equation for conservation of density including $\nabla \cdot \vec{u}$ in order to describe the compressiblity of the fluid. Ignoring compressibility makes impossible to simulate some phenomena, for example those produced by high velocity pressure waves, like sound, so this kind of effects are out of the scope of this project.

Chapter 3

Numerical simulation

3.1 Splitting

Once we know how the Navier-Stokes equations work, we can see how these equations can be solved numerically. To do this, we are going to use a technique called *Splitting* that involves solving the equation in various steps, using as input of each step the output of the previous one. Our aim is to solve the Euler equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \vec{g} - \frac{1}{\rho}\nabla p \tag{3.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{3.2}$$

We have three different steps that will be solved using different numerical techniques and will be applied serially: the input of each step is the output of the previous step:

$$\vec{u}_1 = advect(\vec{u}^n)$$

$$\vec{u}_2 = externalForces(\vec{u}_1)$$

$$\vec{u}^{n+1} = project(\vec{u}_2)$$

The first step will be to solve the convective transport equation:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = 0 \tag{3.3}$$

What we are doing when we solve this equation, from the physics point of view, is to calculate the change in the velocity field due to the motion of the fluid itself. The way this equation is solved is important as it will determine the amount of numerical dissipation the fluid will have. For that reason, different techniques have been compared in this work. In the next section we will see these techniques in detail.

Next step is applying external forces, in this case only gravity:

$$\frac{\partial \vec{u}}{\partial t} = \vec{g} \tag{3.4}$$

This step is trivially solved with Euler explicit integration:

$$\vec{u}_2 = \vec{u}_1 + \Delta t \ \vec{g} \tag{3.5}$$

The last step, called projection is not only the most complex but also the most expensive computationally. This step involves solving the remaining term of the Euler equation:

$$\frac{\partial \vec{u}}{\partial t} = -\frac{1}{\rho}\nabla p \tag{3.6}$$

But, and here is the important part, the resulting velocity field must satisfy the incompressibility $\nabla \cdot \vec{u} = 0$ and boundary conditions. This step will be explained in detail in 3.6 but, briefly, it will involve assembling a system of equations with the pressure values as unknowns and imposing boundary conditions, to the pressure values (Dirichlet) or its normal derivative (Neumman) and imposing also the incompressibility condition. During the simulation, this sequence of operations will be repeated one or more times if necessary for every time step. It is important to note that the projection step must be done before the advection step, because the advection must be done with an incompressible velocity field, i.e. with zero divergence. If it were not the case, the velocity would be advected incorrectly and the fluid would not preserve a constant volume.

3.2 Staggered MAC Grid

First of all, let's see how the simulation domain is discretized in space. Instead of using a standard grid, we are going to use a staggered grid introduced by Harlow and Welch in the 60's in their work about the Marker-and-Cell method for incompressible fluids. This discretization, named MAC grid after the Marker-and-Cell method, consists of storing the horizontal and vertical components of velocity separately so that each component is sampling valocity in a different location in space. Specifically, the horizontal component of velocity will be sampling velocity in the vertical edges of the grid and the vertical component will be sampling velocity in the horizontal edges of the grid. A diagram of this structure for two and three dimensions is shown in figure 3.1: The reason for using this peculiar discretization is that it allows to calculate high precision central differences. In an ordinary grid, with the velocity sampled at the center of each cell, the partial derivatives could be calculated in several ways:

$$\left(\frac{\partial \vec{u}}{\partial x}\right)_i \approx \frac{u_{i+1} - u_i}{\Delta x} \tag{3.7}$$

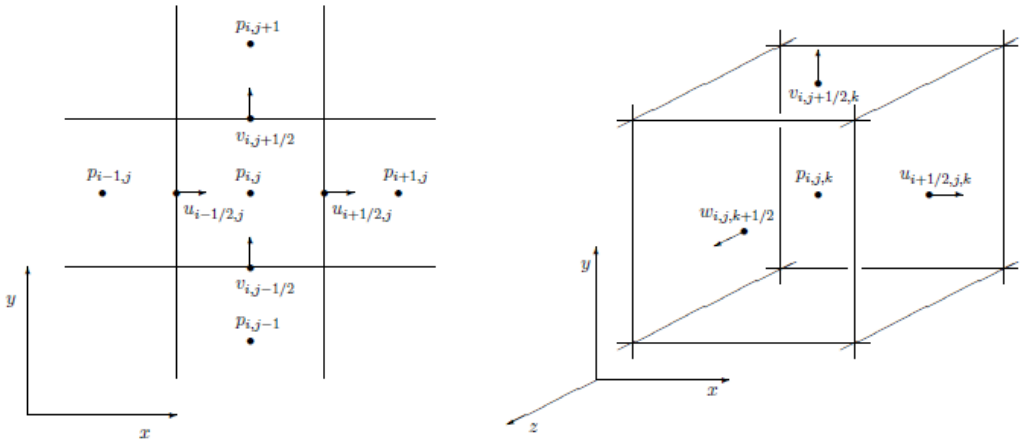


Figure 3.1: MAC Grid

$$\left(\frac{\partial \vec{u}}{\partial x}\right)_i \approx \frac{u_i - u_{i-1}}{\Delta x} \tag{3.8}$$

$$\left(\frac{\partial \vec{u}}{\partial x}\right)_i \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{3.9}$$

The first two equations are forward and backward differences respectively. They have an error of $O(\delta x)$ but are biased in the forward or backward direction. The third one is a central difference with error $O(\delta x^2)$ and unbiased. Central difference could look like the best choice because it is the most accurate, but it has a problem, however. Ignoring the central value u_i whose partial derivative is being calculated, could result in a false zero derivative if the values of u_{i+1} and u_{i-1} are the same. This is called a non-trivial null space. With the spatial discretization defined in the MAC Grid it is possible to have central differences with error $O(\delta x^2)$ and without the null space problem, because no grid value is ignored in the central difference formula:

$$\left(\frac{\partial \vec{u}}{\partial x}\right)_i \approx \frac{u_{i+\frac{1}{2}} - u_{1-\frac{1}{2}}}{\Delta x} \tag{3.10}$$

Using the MAC grid there is a price to pay though, since it is necessary to use more memory to store the additional velocity samples. Besides, the implementation of the numerical methods becomes a little more tricky because it is necessary to perform bi-linear or trilinear interpolations separately for each component in order to get the total velocity in any point inside the grid.

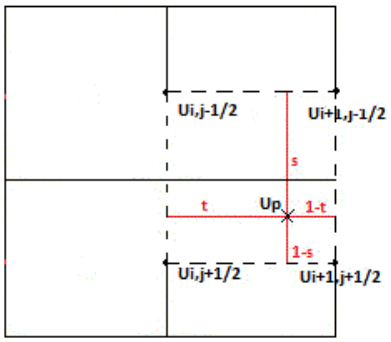


Figure 3.2: Bilinear Interpolation

Figure 3.2 shows how to calculate bilinear interpolation for the u component in the MAC grid, where u_p is the velocity to calculate and is equal to:

$$u_p = (1-s)\left[(1-t)\,u_{i,j-\frac{1}{2}} + t u_{i+1,j-\frac{1}{2}}\right] + s\left[(1-t)\,u_{i,j+\frac{1}{2}} + t u_{i+1,j+\frac{1}{2}}\right] \tag{3.11}$$

It is necessary to establish a correspondence between the MAC Grid indices and the indices for the arrays where the velocity components are stored:

$$u[i][j][k] = u_{i-\frac{1}{2},j,k} \tag{3.12}$$

$$v[i][j][k] = v_{i,j-\frac{1}{2},k} \tag{3.13}$$

$$w[i][j][k] = w_{i,j,k-\frac{1}{2}} \tag{3.14}$$

As it has been said before, with this structure it is necessary to perform a bilinear (2D) or trilinear (3D) interpolation in order to get the value of the velocity vector in an arbitrary point within the domain. However, there are some points like, for example, the center of each cell, where the calculation is very simple because it is only necessary to calculate the average of some grid values:

$$\vec{u}_{i,j,k} = \left(\frac{u_{i+\frac{1}{2},j,k} + u_{i-\frac{1}{2},j,k}}{2}, \frac{v_{i,j+\frac{1}{2},k} + v_{i,j-\frac{1}{2},k}}{2}, \frac{w_{i,j,k+\frac{1}{2}} + w_{i,j,k-\frac{1}{2}}}{2}\right) \tag{3.15}$$

3.3 Semi-Lagrangian advection

As we already know, one of the steps to be solved during the simulation is the advection step:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = 0 \tag{3.16}$$

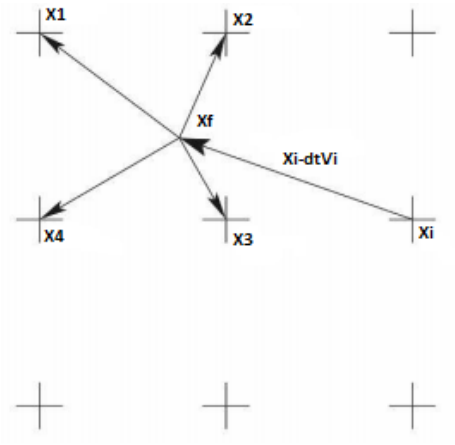


Figure 3.3: Semi-Lagrangian Advection

This transport equation could be solved discretizing in time and solving this equation:

$$u_{i,j,k}^{t1} = u_{i,j,k}^{t} - \Delta t \left[(u \cdot \nabla) \, u \right]^{t} \tag{3.17}$$

The problem with this discretization is that it is conditionally stable, forcing to use a small time step which makes necessary to do more simulation iterations per animation frame increasing considerably the execution time. In 1999 Jos Stam [Sta99] presented a method called "Semi-Lagrangian Advection" that solves the transport equation without imposing any restriction on the time step size. This is how this method works in a nutshell: starting from a grid position whose velocity we want to update, we integrate back in time in order to find the point of the fluid that will occupy the position we want to update. Once we find that point, we get the velocity by interpolating the MAC Grid values and use this value to update the grid position we started from.

Let's put it in equations in order to see it more clearly. We want to get the velocity vector for the next time step $\vec{u_1}$ in the position $\vec{x_I}$. We calculate the position resulting from integrating back in time using the current velocity sampled at $\vec{x_I}$: $\vec{x_F}=\vec{x_I} - \Delta t \, \vec{u^t}$. After that, using bilinear or trilinear intepolation we get the velocity $\vec{u_F}$ at point $\vec{x_F}$. We assign then this velocity as the new velocity at the starting position: $\vec{u_1}=\vec{u_F}$. Figure 3.3 illustrates the technique. In the implementation, a Ruge-Kutta method is used instead of explicit Euler for the backward integration.

The reason for this method to be inconditionally stable is clear: the velocity is always updated with a value obtained interpolating other velocity values of the grid, so the new velocity cannot be bigger than the velocities used in the interpolation. This guarantees that the velocity will never grow up uncontrolled.

3.3.1 Numerical dissipation

The same property that makes the Semi-Lagrangian Advection method so stable, is the cause of its main drawback. In the implementation of Semi-Lagrangian Advection two interpolations are performed for each velocity sample to update. The first one is done to get the velocity vector in the grid point we want to update. The second one is done after integrating back and is done in order to get the new velocity. These two interpolations have the effect of smoothing out the velocity, as if some of the energy had been dissipated, causing the same effect as if the fluid had in fact some viscosity. This side effect is called artificial viscosity and in general is not desirable because it eliminates the high frequencies in the velocity field thus eliminating some small scale effects in the fluid motion.

There are methods that try to avoid this artificial viscosity, for example interpolating with splines instead of linearly. If, for example, the interpolations were done with Catmull-Rom splines this would allow the resulting interpolated velocities to be out of the limits imposed by the source velocities and thus avoiding an excesive smoothing. Another method is the one known as vorticity confinement that involves adding artificial rotational forces to counteract the dissipation.

3.3.2 Boundary conditions

When implementing the Semi-Lagrangian Advection method, it is necessary to define what to do when the integration back in time leads to a point not occupied by a fluid or outside the domain. When this happens, the solution taken is to take the value of the velocity extrapolated, for example taking the velocity of the nearest fluid point.

3.3.3 Timestep size

Although the Semi-Lagrangian Advection method is unconditionally stable, taking a too big timestep could provoke some undesired side effects that could throw back the sense of realism of the fluid animation. Foster and Fedkiw [FN01] suggested to limit the timestep size so that the maximum distance covered by the advection in one timestep was five grid cells:

$$\Delta t \leq \frac{5\Delta x}{u_{max}} \tag{3.18}$$

Where u_{max} is an estimate of the maximum fluid velocity. This estimate could just be the maximum velocity stored in the grid, however Bridson [R.08] suggested that taking u_{max} as:

$$u_{max} = max\left(|u^n|\right) + \Delta t\,|g| \tag{3.19}$$

gives a more accurate estimate that gives better results.

3.4 MacCormack Advection

Semi-Lagrangian Advection traces back a straight line and uses trilinear interpolation to estimate the value. It is first order accurate both in space and time. MacCormack advection [A.07] scheme pushes accuracy to second order while mantaining the unconditional stability of Semi-Lagrangian Advection. This scheme is based on the back and forth error compensation and correction (BFECC) method, so we will start by explaining it briefly. What BFECC does is to estimate the error of the advection method (SLA in this case) by performing an inverse advection of the result and comparing it to the original data sample. This error is used to correct the data and then a third advection is done with the corrected data as input. Let's see it step by step:

1. $\hat{u^{n+1}} = A(u^n)$
2. $\hat{u^n} = A^R(\hat{u^{n+1}})$
3. $error = (\hat{u^n} - u^n)/2$
4. $\bar{u^n} = u^n - error$
5. $u^{n+1} = A(\bar{u^n})$

A() is the advection method. We use SLA, but a different advection method can be used in both BFECC and MacCormack schemes. The difference between the reverse advected velocity calculated in step 2 and the original sample is approximately twice the advection error so, in step 3 the error is estimated dividing that difference by two. The data is corrected in step 4 and finally in step 5 the advection of the corrected data is calculated obtaining the final advected result.

MacCormack uses the same approach but it reduces the number of advection steps to two:

1. $\hat{u^{n+1}} = A(u^n)$
2. $\hat{u^n} = A^R(\hat{u^{n+1}})$
3. $error = (\hat{u^n} - u^n)/2$
4. $u^{n+1} = \hat{u^{n+1}} - error$

The error is calculated in the same way as BFECC, but instead of subtracting it from the original data, it is subtracted from the data advected in the first step giving the final advected result. Figure 3.4 shows the process. ϕ is used instead of u indicating that any quatity can be advected. In Selle et al. [A.07] it is shown that the error of this method is of order $O(\Delta t^3)$, the same as second order Runge-Kutta, and it is shown that it improves significantly the advection of level sets compared to a first order advection

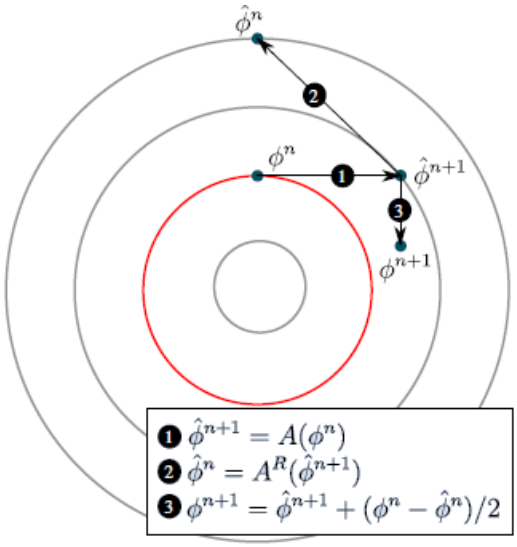


Figure 3.4: MacCormack Scheme

method. It is also noted that these methods (BFECC and MacCormack) do not perform well when information is mixed from outside and inside the boundaries of the fluid, so when the first advection falls out of the fluid boundaries it is recommended to revert to the first order advection method (like SLA)

Although we have been talking about advecting velocity, it is obvious that this method can be used to advect any other quantity.

3.5 PIC/FLIP

PIC and FLIP belong to a family of methods called "hybrid", because they make use of both particles and a fixed grid for solving the fluid equations. PIC was first introduced by Harlow in 1963 [H.86]. It intends to take advantage of both the particle and the fixed grid approaches by using them to solve the tasks they are better suited for: particles for the advection, and fixed grid for everything else. This is how Zhu and Bridson [ZY05] adapted PIC to solve an incompressible flow:

- Initialization of position and velocity in each particle.
- For each simulation time step
  - Transfer velocities from particles to grid
  - Apply external forces on grid
  - Perform projection step on grid
  - Transfer velocities from grid to particles

  - Advect particles using the grid velocity making sure they don't move through solid wall boundaries

The transfer of velocities between particles and grid involves interpolating several times. This causes the PIC method to have an excessive numerical diffusion with the consequent loss of small scale fluid motion.

Brackbill and Ruppel [BJU86] solved the artificial diffusion problem with the FLIP method. The major change was to consider the particles the main representation of the fluid, making use of the grid only to calculate an increment of the particle stored quantities. In the algorithm explained above, this means that velocities are no longer transferred from grid to particles. Instead, the old grid value is subtracted to the new one in order to get the change between the current time step and the previous. This change is then interpolated and added to the particles. This modification on the PIC algorithm reduces considerably the numerical dissipation thus preserving the small scale features of the fluid. The FLIP algorithm would be:

- Initialization of position and velocity in each particle.
- For each simulation time step
  - Transfer velocities from particles to grid
  - Save grid velocities for latter subtraction
  - Apply external forces on grid
  - Perform projection step on grid
  - Subtract the saved velocities from the new ones stored in the grid.
  - Add the interpolated difference to each particle.
  - Advect particles using the grid velocity making sure they don't move through solid wall boundaries

The implementation is so similar that both velocities can be calculated (PIC velocity and FLIP velocity) and an interpolated combination of them can be applied, obtaning a way to control the numerical dissipation in case it is desired. It is important to note that the particle advection is implemented using a more accurate scheme than the one used in the other two methods: instead of only one advection step, five advection subteps are taken checking after each substep that the particles do not cross the boundaries.

3.5.1 Velocity transfer

The velocity transfer from the grid to the particles is quite straightforward since it is just an interpolation of the four (2D) or eight (3D) nearest grid values, separately for each component since we are working with a MAC grid.

In the case of the transfer from the particles to the grid, for each grid point it is calculated a weighted average of the particles contained in the square (2D) or cube (3D) of grid cell width centered on the grid point. The weight of each particle is given by the standard bilinear or trilinear weighting.

3.5.2 Velocity extrapolation

In order to advect the particles using the grid velocities, it is necessary to extrapolate the velocities in the fluid grid cells to the rest of the grid, so that the velocity is defined everywhere. This extrapolation is done solving the equation $\nabla u \cdot \nabla \phi = 0$ where ϕ is a signed distance function [ZY05]. The signed distance function is the distance to the closest point on the fluid surface. It is positive for points outside the fluid and negative for points inside. If one thinks about this function, it is easy to realize that its gradient lines are parallel to the surface normals sice the closest point to a surface is in its normal direction. Also, since the value of this function is the distance to the surface we have that $|| \nabla \phi || = 1$, in fact, we have that $\nabla \phi = \hat{n}$ where $\hat{n}$ is the unit length surface normal. In order to extrapolate the velocity field, it is necessary to construct this distance function in every time step, so a method for solving the equation $|| \nabla \phi || = 1$ (Eikonal equation) is needed. As suggested in [ZY05], the method used is Fast Sweeping [Zha05]. Once the distance function is obtained, the equation $\nabla u \cdot \nabla \phi = 0$ is solved using the fast sweeping method again. This velocity extrapolation step must be performed before advecting the particles.

3.6 Projection

This is the most complex and computationally expensive step. As it was outlined in 3.1, this step solves the pressure term:

$$\frac{\partial \vec{u}}{\partial t} = -\frac{1}{\rho} \nabla p \tag{3.20}$$

producing a velocity field that satisfies the incompressibility condition:

$$\nabla \cdot \vec{u} = 0 \tag{3.21}$$

and also the boundary conditions. Let's begin defining the boundary conditions.

3.6.1 Boundary conditions

The fluid occupies a finite volume and it is necessary to define how the fluid properties are affected in the interface with other media like air and solid in this case. In the case of a solid, we want the fluid not to break through the solid but to flow along it in the

tangential direction. One way to do this is to force the perpendicular component of the velocity to be zero:

$$\vec{u} \cdot \vec{n} = 0 \tag{3.22}$$

This is called 'no-stick' condition, because it allows the fluid to flow along the solid surface. If the solid were moving, then the same condition would be expressed like this:

$$\vec{u} \cdot \vec{n} = \vec{u_{solid}} \cdot \vec{n} \tag{3.23}$$

The normal component of the fluid velocity equals to the same component of the solid's velocity. In [CN11] they use the following inequation instead:

$$\vec{u} \cdot \vec{n} \geq \vec{u_{solid}} \cdot \vec{n} \tag{3.24}$$

This inequation allows the fluid to flow away from the wall preventing stickiness in the normal direction. If we wanted to simulate a viscous fluid accurately, it would be necessary to use a different model that takes into account the adhesion of the fluid particles to the solid boundary, like the no-slip condition which, besides restricting the normal velocity, it also imposes some restriction on the tangential velocity. We will use the no-stick condition as it is easy to implement and gives results good enough for our purposes.

Regarding pressure, it must be taken into account that the pressure not only has to keep the fluid incompressible, but also make the resulting velocity field satisfy the 'no-stick' condition. In the next section we will see how to do this, but briefly it involves calculating 'ghost' pressure values in the cells occupied by solid. These 'ghost' values will be calculated imposing a Neumann condition, or what is the same, a condition depending on $\frac{\partial p}{\partial \vec{n}}$.

In the case of fluid/air interface, the desired behaviour is that the fluid can flow freely acroos the air. To achieve that, the grid cells filled with air will have zero pressure and no restriction will be imposed to the velocity. This definition does not take into account the surface tension. Even though surface tension will not be studied in depth, let's make a brief review of idea behind the simulation of this effect. Surface tension occurs as a consequence of the fact that the attraction force between water molecules is bigger than the attraction force between water and air molecules. In geometric terms this means that the water surface tends to minimize the surface curvature. Hence, what should be done in order to simulate this effect is to apply a pressure in the fluid cells adjacent to the air in such a way that the resulting curvature is minimum. The pressure applied would therefore depend on the surface curvature, that can be calculated with the divergence of the vector normal to the fluid surface.

3.6.2 Assembly

In order to solve the projection step, it is necessary to discretize equations 3.6 and 2.2 and build a system of equations, using equation 3.22 to obtain the pressure values

in the boundary. Let's discretize equation 3.6:

$$u^{n+1}_{i+1/2,j,k} = u_{i+1/2,j,k} - \Delta t \frac{1}{\rho} \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x} \tag{3.25}$$

$$v^{n+1}_{i,j+1/2,k} = v_{i,j+1/2,k} - \Delta t \frac{1}{\rho} \frac{p_{i,j+1,k} - p_{i,j,k}}{\Delta x} \tag{3.26}$$

$$w^{n+1}_{i,j,k+1/2} = w_{i,j,k+1/2} - \Delta t \frac{1}{\rho} \frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta x} \tag{3.27}$$

As it can be seen, one equation is posed for each velocity component, discretizing in each case the pressure gradient in the corresponding direction. Let's go back to the 'no-stick' condition: $\vec{u} \cdot \vec{n} = \vec{u_{solid}} \cdot \vec{n}$. We will assume that when a cell contains a solid, the solid always occupies the whole cell, i.e. the boundary of the solids will always be aligned with the x,y and z axis. The 'no-stick' condition then states that, when a cell of fluid is adjacent to a solid cell, one of the three componentes of the velocity of the fluid cell must be zero after the projection step. Which component will be zero, depends on where the solid cell is with respect to the fluid cell, for example, if the solid cell would be in the previous cell in the x direction, we would have $u^{n+1}_{i+1/2,j,k} = u_{solid}$ and replacing this in equation 3.25, we would get this value for the pressure in the solid cell:

$$p_{i+1,j,k} = p_{i,j,k} + \frac{\rho \Delta x}{\Delta t} \left(u_{1+1/2,j,k} - u_{solid} \right) \tag{3.28}$$

It is important to note that, since this calculation is done per component, if a solid cell is surrounded by more than one fluid cell, this cell will have more than one pressure value associated with it. This may seem strange, but remember that these pressure values have no physical meaning, they are calculated only with the purpose of avoiding the fluid to go through the solid walls.

Let's discretize equation 2.2. Expanding the divergence operator for three dimensions we have:

$$\nabla \cdot \vec{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{3.29}$$

With the MAC Grid, we can use the accurate central differences to discretize equation 2.2 and then we get the incompressibility condition for the resulting velocity:

$$\frac{u^{n+1}_{i+1/2,j,k} - u^{n+1}_{i-1/2,j,k}}{\Delta x} + \frac{v^{n+1}_{i,j+1/2,k} - v^{n+1}_{i,j-1/2,k}}{\Delta x} + \frac{w^{n+1}_{i,j,k+1/2} - w^{n+1}_{i,j,k-1/2}}{\Delta x} = 0 \tag{3.30}$$

Putting all this together, we can now pose a system of equations. We start substituting velocities from equations 3.25, 3.26 and 3.27 as appropriate in equation 3.30. This is

the result:

$$\frac{1}{\Delta x}\left[\left(u_{i+1/2,j,k}-\Delta t\frac{1}{\rho}\frac{p_{i+1,j,k}-p_{i,j,k}}{\Delta x}\right)-\left(u_{i-1/2,j,k}-\Delta t\frac{1}{\rho}\frac{p_{i,j,k}-p_{i-1,j,k}}{\Delta x}\right)+\left(v_{i,j+1/2,k}-\Delta t\frac{1}{\rho}\frac{p_{i,j+1,k}-p_{i,j,k}}{\Delta x}\right)-\left(v_{i,j-1/2,k}-\Delta t\frac{1}{\rho}\frac{p_{i,j,k}-p_{i,j-1,k}}{\Delta x}\right)+\left(w_{i,j,k+1/2}-\Delta t\frac{1}{\rho}\frac{p_{i,j,k+1}-p_{i,j,k}}{\Delta x}\right)-\left(w_{i,j,k-1/2}-\Delta t\frac{1}{\rho}\frac{p_{i,j,k}-p_{i,j,k-1}}{\Delta x}\right)\right]=0 \quad (3.31)$$

Grouping the pressure terms in the left hand side and velocities in the right hand side we have:

$$\frac{\Delta t}{\rho}\left(\frac{6p_{i,j,k}-p_{i+1,j,k}-p_{i,j+1,k}-p_{i,j,k+1}-p_{i-1,j,k}-p_{i,j-1,k}-p_{i,j,k-1}}{\Delta x^2}\right)=-\left(\frac{u_{i+1/2,j,k}-u_{i-1/2,j,k}}{\Delta x}+\frac{v_{i,j+1/2,k}-v_{i,j-1/2,k}}{\Delta x}+\frac{w_{i,j,k+1/2}-w_{i,j,k-1/2}}{\Delta x}\right) \quad (3.32)$$

Equation 3.32 turns to be the discretized version of the Poisson equation:

$$\frac{\Delta t}{\rho}\nabla^2 p=-\nabla\cdot u \quad (3.33)$$

Let's see an example of how to integrate boundary conditions in this system of equations. Suppose the cell $(i,j,k+1)$ is an air cell and cell $(i+1,j,k)$ is a solid cell. The equation would then be:

$$\frac{\Delta t}{\rho}\left(\frac{6p_{i,j,k}-\left(p_{i,j,k}+\frac{\rho\Delta x}{\Delta t}\left(u_{1+1/2,j,k}-u_{solid}\right)\right)-p_{i,j+1,k}-0-p_{i-1,j,k}-p_{i,j-1,k}-p_{i,j,k-1}}{\Delta x^2}\right)=-\left(\frac{u_{i+1/2,j,k}-u_{i-1/2,j,k}}{\Delta x}+\frac{v_{i,j+1/2,k}-v_{i,j-1/2,k}}{\Delta x}+\frac{w_{i,j,k+1/2}-w_{i,j,k-1/2}}{\Delta x}\right) \quad (3.34)$$

Grouping terms, the coefficient of $p_{i,j,k}$ is reduced to 5 and the term $\frac{\rho\Delta x}{\Delta t}\left(u_{1+1/2,j,k}-u_{solid}\right)$ is taken out after eliminating $\frac{\rho}{\Delta t}$ with $\frac{\Delta t}{\rho}$ and dividing Δx by Δx^2. Moving this term to the right hand side, the result is:

$$\frac{\Delta t}{\rho}\left(\frac{5p_{i,j,k}-p_{i,j+1,k}-p_{i-1,j,k}-p_{i,j-1,k}-p_{i,j,k-1}}{\Delta x^2}\right)=-\left(\frac{u_{i+1/2,j,k}-u_{i-1/2,j,k}}{\Delta x}+\frac{v_{i,j+1/2,k}-v_{i,j-1/2,k}}{\Delta x}+\frac{w_{i,j,k+1/2}-w_{i,j,k-1/2}}{\Delta x}\right)+\left(\frac{u_{1+1/2,j,k}-u_{solid}}{\Delta t}\right) \quad (3.35)$$

Finally, solving the right hand side we get:

$$\frac{\Delta t}{\rho}\left(\frac{5p_{i,j,k} - p_{i,j+1,k} - p_{i-1,j,k} - p_{i,j-1,k} - p_{i,j,k-1}}{\Delta x^2}\right) = \\ -\left(\frac{u_{solid} - u_{i-1/2,j,k}}{\Delta x} + \frac{v_{i,j+1/2,k} - v_{i,j-1/2,k}}{\Delta x} + \frac{w_{i,j,k+1/2} - w_{i,j,k-1/2}}{\Delta x}\right) \quad (3.36)$$

Looking carefully at this result, a general rule can be derived. For each adjacent air cell, the corresponding term in the left hand side is removed. For each adjacent solid cell, the corresponding pressure term is removed, the coefficient of $p_{i,j,k}$ is reduced by one and the corresponding velocity term in the right hand side of the equation is replaced with u_{solid}. Since the coefficient of $p_{i,j,k}$ is reduced by one for each adjacent solid cell, this coefficient is in fact equal to the number of adjacent non solid cells.

Once we have seen how to pose the Poisson equation for each cell, we can assemble all the equations in a matrix system:

$$Ap = b \quad (3.37)$$

Where p is a vector of size equal to the number of grid cells and b is a vector with the same size containing the velocity divergence of each cell. A is a matrix whose elements are integer numbers corresponding to the coefficients of the pressures and it will be multiplied by $\frac{\Delta t}{\rho \Delta x^2}$. Let's analyze in detail the structure of the matrix A.

A has a size of $N \times N$ where N is the number of cells in the grid. Each row in A corresponds to a Poisson equation like 3.32 The rows corresponding to non fluid cells will be set to zero, since the velocity does not need to be calculated in these cells. The rows corresponding to fluid cells, will have the coefficient of $p_{i,j,k}$ in the cell located in the matrix diagonal. As we saw earlier, this coefficient will be equal to the number of non solid cells adjacent to the cell (i, j, k), so it will be at most six in three dimensions. In the rest of cell of the row, the value will be -1 if the corresponding adjacent cell is fluid or 0 if it is solid, air, or if the corresponding cell is not adjacent. Summarizing, each row will be either all zeros, or a number from 0 to 6 in the diagonal and up to six cells with -1 depending on the number of fluid adjacent cells. Looking at the matrix structure carefully, we find that the matrix is symmetric, e.g., the element $A_{(i,j,k)(i+1,j,k)}$ will be equal to $A_{(i+1,j,k)(i,j,k)}$ because this elements will be zero if any of the cells is not fluid or -1 if both cells are fluid. Besides being symmetric, this matrix is also positive definite, a property that will be important when choosing the appropriate method for solving the system of equations, as will be seen in the next section.

Another interesting property of A is that it is a sparse matrix, i.e., its elements are mostly zeros. This property, in addition to the fact that the matrix is symmetric, makes it possible to omit the storage of most of the elements of the matrix. For each matrix row, it is only necessary to store the coefficient of $p_{i,j,k}$ and, thanks to the symmetry, three of the six elements corresponding to the adjacent cells. Following Bridson's [R.08]

notation, the coefficients of $p_{i,j,k}$ will be stored in an array named $Adiag(i,j,k)$ whose size will be equal to the number of cells in the grid (N). The three remaining elements per row will be stored in arrays of size N which we will call $Aplusi(i,j,k)$, $Aplusj(i,j,k)$ and $Aplusk(i,j,k)$. With this structure, if one wants to read the element $A_{(i,j,k)(i,j-1,k)}$ of the matrix, for example, due to symmetry it is sufficient to read the element $A_{(i,j-1,k)(i,j,k)}$ which is stored in the array $Aplusj$ in the position $Aplusj(i,j-1,k)$. Once the system of equations is defined, it is time to look for the appropriate method to solve it.

3.6.3 Conjugate Gradient

The method that has been selected to solve the system of equations presented in the last section is a method called "Modified Incomplete Cholesky Conjugate Gradient, Level Zero" as explained in [R.08]. This method is a modification of the Conjugate Gradient algorithm, so let's start explaining this one before going into detail with the complete algorithm.

As it was noted before, the matrix A is symmetric and positive definite. Conjugate Gradient is an iterative algorithm which is used to solve systems of equations with symmetric and positive definite matrices. Being iterative, the algorithm starts with an estimation of the solution and it improves it with each iteration until a minimum precision is achieved. A great benefit of this algoritm is that the operations performed in each iteration are matrix-vector products, vector-scalar products and vector additions, operations that can be implemented very efficiently. Each iteration of the method calculates the "residual vector":

$$r_i = b - Ap_i \tag{3.38}$$

This vector is the difference between the left-hand side and the right-hand side of the equation, where the vector p_i is the approximated solution at iteration i. If p_i were an exact solution, the vector r_i should be zero, so we can use the infinity norm of r_i to estimate the error of the solution p_i and therefore define a stopping condition when this error is under a given threshold. The infinity norm of a vector is the element whose absolute value is the largest. The threshold used as stopping condition has to be chosen in a way that nor is it too small so the compute time is short enough or too big so the solution is not accurate enough and leads to an unattractive animation. Besides the error threshold, it is necessary to define a maximum number of iterations to avoid the algorithm to take too much time to converge due to numerical errors because of the floating point format precision errors. In [R.08], it is recommended to start with a threshold of 10^{-6} and 100 iterations.

The algorithm used is a variation of Conjugate Gradient (CG) called Preconditioned Conjugate Gradient (PCG) which has the benefit of converging faster and consequently needs less iterations to achieve the solution with the desired precision. The idea behind preconditioning starts by considering that the solution of $Ap = b$ is the same as the solution of $MAp = Mb$ where M is some matrix. Now, if M is approximately A^{-1} so

MA is very close to the identity matrix, then the Conjugate Gradient algorithm should solve the preconditioned system of equations $MAp = Mb$ very quickly.

The preconditioner which we will use here is based on the Cholesky factorization. Cholesky factorization has the form $A = LL^T$ where A is symmetric positive definite and L is a triangular matrix. When solving a system of equations of the form $Ap = b$, if A is suitable for Cholesky factorization, then the system can be solved doing a forward and a backward substitutions:

$$Ap = b$$
$$LL^T p = b$$
$$Lq = b$$
$$L^T p = q$$

The aim of Incomplete Cholesky is to calculate a matrix L such that solving with forward and backward substitution is close to applying A^{-1} as we saw before. The problem with Cholesky factorization is that, although A is a sparse matrix, L can have a lot of non zero elements, so it is not suitable for our solver as it would involve too much memory usage. What Incomplete Cholesky does is to calculate the matrix L following the original algorithm but putting zeros in all the elements of L when the corresponding element in A is zero, thus preserving the same number of non zero elements as in A. Now suppose that A is split up into triangle and diagonal matrices:

$$A = F + D + F^T \tag{3.39}$$

Then it turns out that

$$L = FE^{-1} + E \tag{3.40}$$

Where E is a diagonal matrix. This result is not going to be demonstrated here, so we will just assume it is true. Since F is just the triangular part of A, this result implies that the calculation of L only involves storing the diagonal of E, wich significantly reduces the memory cost. In three dimensions, the diagonal matrix E is calculated with this formula:

$$E_{(i,j,k)} = \sqrt{\begin{array}{l} A_{(i,j,k)(i,j,k)} - \left(A_{(i-1,j,k)(i,j,k)}/E_{(i-1,j,k)}\right)^2 - \left(A_{(i,j-1,k)(i,j,k)}/E_{(i,j-1,k)}\right)^2 - \\ \left(A_{(i,j,k-1)(i,j,k)}/E_{(i,j,k-1)}\right)^2 \end{array}} \tag{3.41}$$

Remind that, if N is the number of cells in the computational domain, E is a vector of size N and A is a matrix of size $N \times N$. The notation (i, j, k) is used to represent a single index corresponding to a single cell of the domain. Note that the calculation of each element of E makes use of other values of E previously calculated. If a term refers to a non fluid cell or a cell outside the domain, it is replaced with zero.

Incomplete Cholesky, IC from now on, is a good preconditioner but we are going to use 'Modified' IC which goes a step further and achieves even better performance. When we calculate IC, we discard every element of L if the corresponding element in

A is zero. In the modified version of IC, these elements are not discarded but added up to the correspondig diagonal element. The resulting matrix then, besides having the same non zero elements outside the diagonal, it also has the property that the sum of each row is equal to the sum of the same row in A. This new definition of L derives in a different calculation for E:

$$E_{(i,j,k)} = \sqrt{\begin{array}{l} A_{(i,j,k)(i,j,k)} - \\ (A_{(i-1,j,k)(i,j,k)}/E_{(i-1,j,k)})^2 - (A_{(i,j-1,k)(i,j,k)}/E_{(i,j-1,k)})^2 - (A_{(i,j,k-1)(i,j,k)}/E_{(i,j,k-1)})^2 - \\ A_{(i-1,j,k)(i,j,k)}(A_{(i-1,j,k)(i-1,j+1,k)} + A_{(i-1,j,k)(i-1,j,k+1)})/E^2_{(i-1,j,k)} - \\ A_{(i,j-1,k)(i,j,k)}(A_{(i,j-1,k)(i+1,j-1,k)} + A_{(i,j-1,k)(i,j-1,k+1)})/E^2_{(i,j-1,k)} - \\ A_{(i,j,k-1)(i,j,k)}(A_{(i,j,k-1)(i+1,j,k-1)} + A_{(i,j,k-1)(i,j+1,k-1)})/E^2_{(i,j,k-1)} \end{array}} \tag{3.42}$$

Something to bear in mind when using PCG is that, the better the initial estimate of pressure is, the quicker the algorithm will converge to an accurate solution. In the case of a relatively stationary fluid, which does not change drastically from one simulation step to the next, it would be a good idea to use the pressure obtained in the previous step as initial estimate for the current step. However, the fluid animations that we usually want to create involve a fluid constantly moving and changing rapidly, so using the previous pressure values carries no benefit. In [R.08], it is recommended to use a zero vector for the initial estimate of pressure as it works reasonably well even for rapidly changing fluids.

Once we know all the details, we can now present the whole algorithm:

- Initialize solution p = 0 and residual vector r = b
- z = preconditioner * r
- s = z
- $\sigma = dot(z, r)$
- Loop (max. iterations)
- z=A*s
- $\alpha = \rho / dot(z, s)$
- $p+ = \alpha s$
- $r- = \alpha z$
- $if\ ||\ r\ ||_\infty <= tolerance then return p$
- z = preconditioner * r
- $\sigma_{new} = dot(z, r)$
- $\beta = \sigma_{new} / \rho$
- $s = z + \beta s$
- $\sigma = \sigma_{new}$
- return p (max. iterations exceeded)

Chapter 4

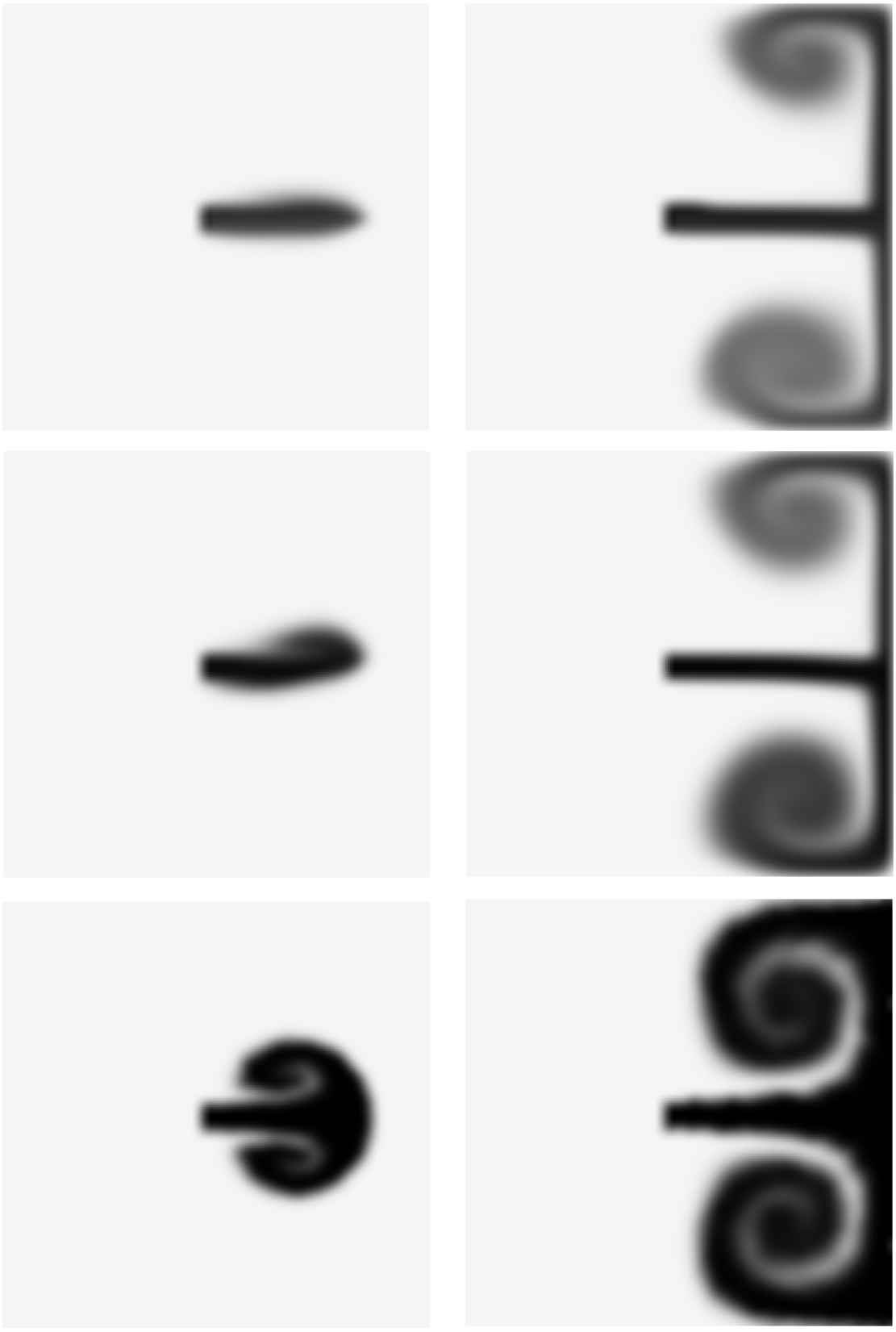
Results

In order to compare the results obtained with the different advection techniques, we have run the same test three times, one per advection method. The test consists of a square filled with fluid, with initial velocity zero in the whole domain. In the first time step, i.e. at $t = \Delta t$, an artificial source of black ink is created in five cells located at the center of the domain with velocity $V_x = 10\,m/s$ and $V_y = 0\,m/s$. The density of the ink is set to 1.0 in the five cells and is then advected with the velocity using the corresponding advection method. A density of 1.0 is painted as black and 0.0 as white. Note that assigning an arbitrary velocity to some cells while all the others have velocity zero creates a discontinuity in the velocity field, but this is not a problem because, after the first time step, the projection step ensures the incompressibility condition smoothing this discontinuity in the velocity field.

In table 4.1 below we can see three pairs of pictures, each pair corresponding to a different advection method. In all three cases the screenshot was taken at the same instants $t = 0.3\,s$ and $t = 1.5\,s$. It can be observed that the better the advection method is, the more it conserves the density field. Recalling the advection methods, what SLA does is to follow the velocity in opposite direction and then perform a bilinear interpolation of the density in the corresponding cells. This interpolation coupled with the interpolations performed in the MAC grid in order to get the velocity causes an artificial dissipation that results in the density fading away and therefore in an apparent loss of volume.

This undesirable effect is partially mitigated by the MacCormack method. This method does a SLA and then an inverse SLA in order to calculate an estimation of the error, and then uses this estimation to correct the first SLA. The pictures show how this slightly improves the conservation of the ink volume. The picture corresponding to $t = 0.3\,s$ shows how a small swirl starts developing whilst in the SLA case the swirl is non-existing. Also, in the pictures corresponding to $t = 1.5\,s$ it can be seen that the swirls in the MacCormack case are bigger and more black than the ones in SLA, meaning that MacCormack preserves better the density field. This difference between MacCormack and SLA becomes larger using bigger time steps.

Table 4.1: Advection Results



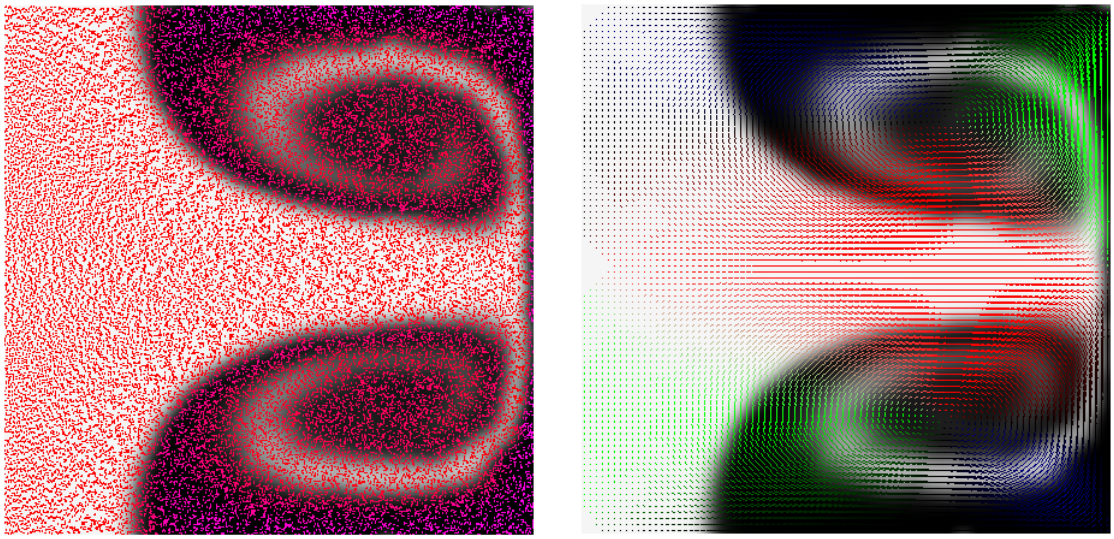
With FLIP, on the other hand, the pictures show that the improvement is dramatic. At $t = 0.3\,s$ the ink shows two symmetrical swirls at both sides with considerably more volume than in the other two cases. In the picture at $t = 1.5\,s$ we can see a fluid with much more volume because FLIP prevents much more efficiently the loss of volume of the density field by reducing the numerical dissipation.

Table 4.2 shows the evolution of the average volume per cell for the three methods. Since we are representing the ink as a density field whose value is $0.0 \leq density \leq 1.0$, we can calculate the average volume by adding up the density of all cells and dividing by the total number of cells ($90 \times 90 = 8100$ cells).

Table 4.2: Average volume per cell

t(s)	0.3	0.6	0.9	1.2	1.5
SLA vol.	0.008	0.015	0.027	0.037	0.048
MacCormack vol.	0.013	0.023	0.040	0.051	0.070
FLIP vol.	0.060	0.103	0.15	0.219	0.272

Table 4.3: FLIP Particles and Velocity



In the initialization of FLIP, four particles are randomly placed in each cell. Every particle is initialized with zero ink density and the density is advected the same way velocity is: the density is transferred from the grid to the particles, and after the particles are advected the density is transferred back to the grid. Table 4.3 shows the particle distribution in the FLIP method and also the velocity field interpolated in the center of the cells. The velocity is represented with different colors depending on the velocity direction. Green is for velocity in y axis, red in x axis and blue in $-y$ axis.

Let's now analyze the ratio domain-size/computation-time of each method. Tables 4.4, 4.5 and 4.6 show the time each method spends in each step, and also the percetage that the most time consuming steps take with respect to the total time. If we take a look at Tables 4.4 and 4.5, they show that both SLA and MacCormack advection methods are computationally cheap compared to the projection step. In fact, as the domain size grows, it can be seen that the projection step takes more and more percentage of the total time. The conclusion here is that if you use SLA or MacCormack schemes, all the optimization efforts should be directed to the projection step.

Table 4.6 on the other hand, shows a different behaviour. For small resolutions, the advection step takes more time than the projection step. To explain this, we need to recall the FLIP advection explained in section 3.5. In that section we noted that the advection step is divided in five substeps. Besides, each fluid cell has an average of four particles (eight in 3D), so the FLIP advection involves $5 * 4 = 20$ Euler integrations per fluid cell while MacCormack performs eight and SLA only four. For that reason, the advection step is the most time consuming step in FLIP with low resolutions. However, the time table shows that the more resolution, the more important the projection step becomes. In fact, with a resolution of 500×500 the projection step takes almost a 40% more than the advection step, and this difference will grow as the resolution increases making the projection step the most time consuming as it is in SLA and MacCormack.

Table 4.4: SLA time table

Resolution	Advection(ms.)	Projection(ms.)	Total(ms.)	Projection%
25x25	0	1	1	100%
50x50	1	4.5	5.5	71%
100x100	5	30	35	84%
150x150	11	90	101	87.5%
200x200	20	210	240	90%

Table 4.5: MacCormack time table

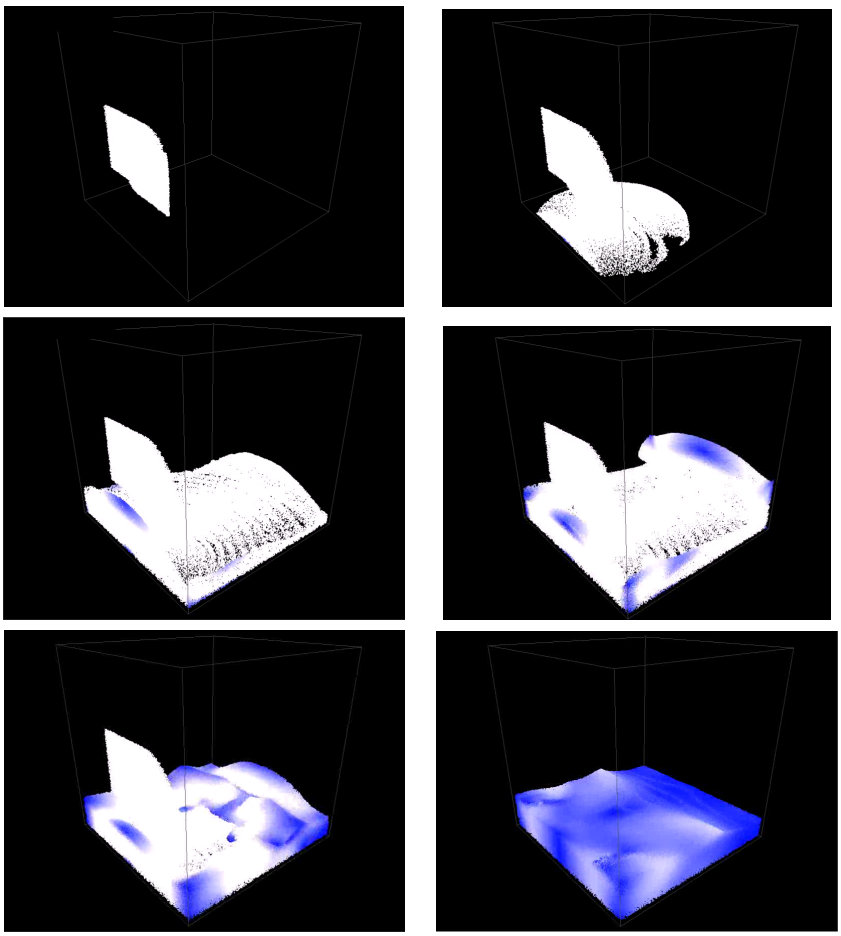
Resolution	Advection(ms.)	Projection(ms.)	Total(ms.)	Projection%
25x25	0	1	1	100%
50x50	2	4.5	6.5	70%
100x100	10	31	41	74%
150x150	22	90	112	78%
200x200	41	207	248	83%

Table 4.6: FLIP time table

Res.	Advect.(ms.)	Transfer(ms.)	Project.(ms.)	Total(ms.)	Advect.%	Projec.%
25x25	4	1	1	6	67%	17%
50x50	16	5	5	26	65%	19%
100x100	63	38	32	133	47%	24%
150x150	142	70	98	310	45%	32%
200x200	249	104	189	542	46%	35%
500x500	1580	587	2150	4317	37%	50%

Finally, table 4.7 shows some frames of a simulation of water being poured in a cubic container. This simulation was done with the full 3D FLIP solver. The quality of the simulations obtained with this solver is remarkable.

Table 4.7: FLIP 3D



Chapter 5

Conclusions and future work

Fluid simulation is a very extensive area of research. There are many different techniques and approaches and new ones are presented each year. In this work we have reviewed and compared some of the most important techniques and developed a source code able to run 3D fluid simulations which serve as a base for the later development of a more complete and efficient fluid simulator based on hybrid techniques and with a more sophisticated rendering method. Here is a summary of some of the researched techniques which could greatly improve the quality of the solver.

5.1 Optimization

5.1.1 Parallelization

All the code in this project runs in the CPU (no CUDA or OpenCL) and in a single-threaded fashion. Some portions of the code could be easily parallelized so drastically reducing the computational cost. For example, the advection step performs the same operation in every cell. This operation involves reading data from various cells and writting in one single cell (gather operation). This is perfectly suited for GPU parallelization and the performance gain would be important.

The Pareto principle states that usually 20% of the code consumes 80% of the resources so, when one wants to optimize their software, the first thing to do is locate that 20% of the code. As we saw in chapter 4, in our fluid solver this 20% of the code is the pressure solve (projection step) This operation could also be parallelized and any performance gain in this step would have a deep impact in the overall performance.

5.1.2 Pressure solve optimization

Besides parallelizing, there are other ways to improve the performance of the projection step. In Lentine et al. [LM10], a new method is presented which reduces the cost of the projection step doing a single projection in a coarse grid, followed by a number of small projections, each one in a small portion of the original grid. Here is an outline of the algorithm:

- Map the data from the finer grid to the coarser one using an area-weighted average:

$$\vec{u}_{coarse} = 1/n \sum_{f \in faces} A_f \vec{u}_f$$

- Solve the projection step in the coarser grid.
- Map the data back to the finer grid:

$$\vec{u}_f^{n+1} = \alpha \vec{u}_c^{n+1} + (1-\alpha)(\vec{u}_f + \vec{u}_c^{n+1} - \vec{u}_c)$$

- Solve the projection step separately for each domain portion in the finer grid.

The algorithm can be combined with an octree-like structure, solving the projections in a hierarchical way. The main advantage of the method is that it improves significantly the scalability of the projection step, The finer projections can be solved independently, making the algorithm well suited for clusters of cores without shared memory.

5.1.3 Tall cells

Tall cells is a recent method presented by Chentanez and Müller [CN11] that combines a multigrid solver with a simplification of the grid cells in regions where no detail is nedeed. Specifically, it groups all cells in a column of a fluid, from the bottom to a point close to the surface, in one single tall cell which contains only two samples of data, one in the top of the cell and the other in the bottom as can be seen in figure 5.1 In their article, Chentanez and Müller [CN11] define specific criteria for determining the appropriate height of each tall cell. This method optimizes all stages of the fluid solver by collapsing all the cells that it considers not important for the fluid motion effect.

5.2 Surface tracking

When a fluid is represented using particles, it is possible to reconstruct the fluid surface from the particles. Zhu and Bridson [ZY05] proposed to define a distance field for each particle and then calculate a weighted average of these scalar fields to get a scalar function representing the fluid surface:

$$\phi(x) = \mid x - \bar{x} \mid - \bar{r}$$

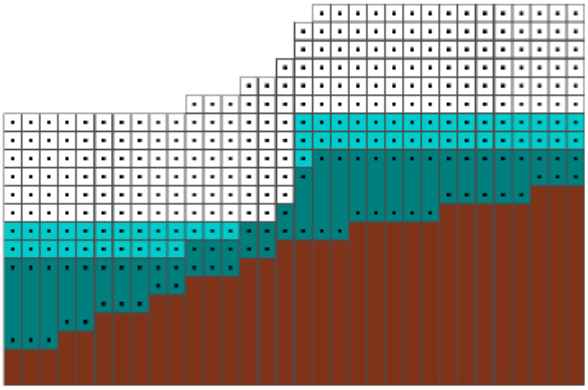


Figure 5.1: Tall Cells

$$\bar{x} = \sum_i w_i x_i$$

$$\bar{r} = \sum_i w_i r_i$$

$$w_i = \frac{k(\mid x - x_i \mid /R)}{\sum_j k(\mid x - x_j \mid /R)}$$

Where k is a smooth kernel function. In [BS07] Solenthaler et al. improved this approach defining the implicit surface function as:

$$\phi(r) = \mid r - \bar{r} \mid - R_f$$

$$\bar{r} = \frac{\sum_j rW(\mid r - r_j \mid, ir)}{\sum_j W(\mid r - r_j \mid, ir)}$$

Where f is a factor $\in$ [0..1], ir is the *influence radius* which defines the smoothness of the surface and W is the density kernel function of the SPH method used by Solenthaler et al [BS07].

These methods generate inevitably some artifacts in the fluid surface. Besides, they require particles in order to construct the surface. Other popular methods which do not have these disadvantages are level set methods. Level Set methods roughly consist on advecting a signed distance that describes the fluid surface in each time step, thus evolving the fluid surface according to its velocity. The main disadvantage of this method is that a very accurate integration method is required in order to obtain good results, and this makes it very expensive computationally. In [FN01] Foster and Fedkiw developed the particle level set method which took advantage of particles to advect the fluid surface. This particle method was later improved by Enright et al by placing particles inside the fluid as well as outside, to accurately measure the volume loss/gain with the advection surface and using this measurement to correct the level set.

5.3 Surface rendering

Fluid rendering is a vast area of research and it will be briefly mentioned here. The most popular methods for fluid rendering tipically start from the base that there is an iso-surface definig the fluid surface. In the case of an Eulerian simulation, this iso-surface is given by a level set, i.e. a signed distance function which is advected with the fluid (there are different methods for this as commented in 5.2). In a particle based simulation on the other hand, the iso-surface can be defined as a density field formed as a superposition of kernel functions of the individual particles (Bridson [R.08], Solenthaler et al. [BS07]). Either way, once the iso-surface is defined, the most typicall methods for rendering it are ray-tracing and marching cubes. Ray-tracing consists of casting rays from the camera to the scene and calculating the intersection of this rays with the fluid iso-surface. Marching cubes is a method whose input is an iso-surface and whose output is a triangular mesh that can be easily rendered with by 3D hardware. None of these methods is suitable for a real-time software like a videogame.

There are some other approches like the one presented in Müller et al. [MM07]. The authors of this method claim that it can be used for real-time applications. This method assumes that the fluid is represented with particles. It calculates the silhouette of the fluid and generates a triangular mesh which is then unprojected back to word space. The output mesh is view dependent so it only makes sense from the camera point of view. This method shows that there are interesting alternatives to the traditional ray-tracing and marching cubes methods.

Appendix A

Material derivative

In 2.1 we presented the Navier-Stokes equations using an operator named "Material Derivative":

$$\frac{D\ \vec{u}}{Dt} = \frac{\partial\ \vec{u}}{\partial t} + \vec{u} \cdot \nabla\ \vec{u} \tag{A.1}$$

The concept of material derivative is very important in fluid simulation as it is the link between the Lagrangian and Eulerian description of the equations. Let's think about a generic quantity q referring to some property of the fluid. Every element of fluid has its value for q and q also changes over time. q is then described by $q\left(t, \vec{x}\right)$ This is an Eulerian description of q since it is a function of position and time. The Lagrangian description tells us how fast is q changing for a particular fluid element. This is obtained by taking the derivative over time of $q\left(t, \vec{x}\right)$:

$$\frac{d}{dt} q\left(t, \vec{x}\right) = \frac{\partial q}{\partial t} + \nabla q \cdot \frac{d\ \vec{x}}{dt} = \frac{\partial q}{\partial t} + \nabla q \cdot \vec{u} \equiv \frac{Dq}{Dt} \tag{A.2}$$

And this is the material derivative. The first term of the material derivative, $\frac{\partial q}{\partial t}$, indicates how q changes over time at that fixed position in space. The other term, $\nabla q \cdot \vec{u}$, adds the change due to fluid flowing with velocity $\vec{u}$ through that position.

Appendix B

Divergence

As we said in section 2.4, this appendix will show that the divergence of the velocity in a smal fluid element is approximately the gain of fluid per unit volume. Let's say we have a fluid element moving with velocity v such as the one shown in figure B.1.

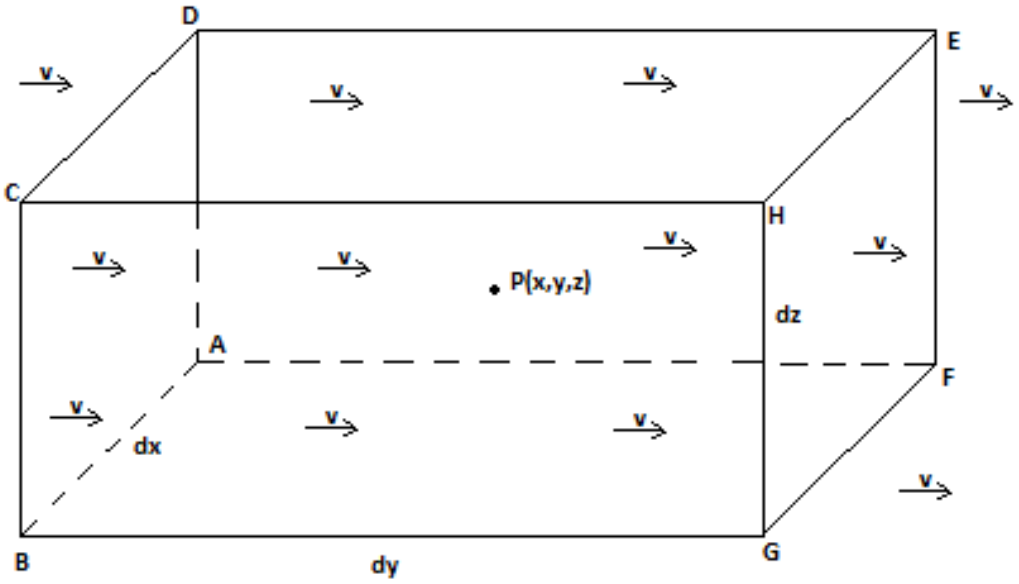


Figure B.1: Fluid flowing through fluid element

The velocity at point P is $v = (v_1, v_2, v_3)$. Using the first two terms of Taylor series, the x component of v at the center of the face $AFED$ is approximately:

$$v_1 - \frac{1}{2}\frac{\partial v_1}{\partial x}\Delta x \tag{B.1}$$

The same can be done for $GHCB$:

$$v_1 + \frac{1}{2}\frac{\partial v_1}{\partial x}\Delta x \tag{B.2}$$

Now, the volume of fluid crossing $AFED$ and $GHCB$ per unit time is:

$$\left(v_1 - \frac{1}{2}\frac{\partial v_1}{\partial x}\Delta x\right)\Delta y\Delta z \tag{B.3}$$

$$\left(v_1 + \frac{1}{2}\frac{\partial v_1}{\partial x}\Delta x\right)\Delta y\Delta z \tag{B.4}$$

The gain of volume is then equation B.3 minus equation B.4:

$$\frac{\partial v_1}{\partial x}\Delta x\Delta y\Delta z \tag{B.5}$$

Similarly, the gain of volume in y and z direction is:

$$\frac{\partial v_2}{\partial y}\Delta x\Delta y\Delta z \tag{B.6}$$

$$\frac{\partial v_3}{\partial z}\Delta x\Delta y\Delta z \tag{B.7}$$

Then, the total gain in volume per unit volume per unit time is:

$$\frac{\left(\frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y} + \frac{\partial v_3}{\partial z}\right)\Delta x\Delta y\Delta z}{\Delta x\Delta y\Delta z} = \nabla \cdot v \tag{B.8}$$

Note that this is true only in the limit as the parallelepiped shrinks to P, i.e. as Δx, Δy and Δz approach zero.

Bibliography

[A.07] Selle A. *An Unconditionally Stable MacCormack Method.* J. Sci. Comput. in review, 2007. 2, 3, 15

[BJU86] Ruppel H. M. Brackbill J. U. *FLIP: a method for adaptively zoned, particle-in-cell calculuations of fluid flows in two dimensions.* J. Comp. Phys, 1986. 2, 17

[BS07] R. Pajarola B. Solenthaler, J. Schläfli. *A Unified Particle Model for Fluid-Solid Interactions.* Computer Animation and Virtual Worlds, 2007. 35, 36

[CN11] Müller M. Chentanez N. *Real-Time Eulerian Water Simulation Using a Restricted Tall Cell Grid.* In Proc. SIGGRAPH, 2011. 2, 19, 34

[D.03] Takeshita D. *Particle-based visual simulation of explosive flames.* Computer Graphics and Applications, 2003. 3

[DE02] Ferziger J. I. Mitchell D. Enright, Fedkiw J. *A hybrid particle level set method for improved interface capturing.* J. Comp. Phys, 2002. 2

[FN96] Metaxas D. Foster N. *Realistic animation of liquids.* Graphical Models and Image Processing, 1996. 2

[FN01] Fedkiw R. Foster N. *Practical animation of liquids.* In Proc. SIGGRAPH, 2001. 2, 14, 35

[H.86] Harlow F. H. *The particle-in-cell method for numerical solution of problems in fluid dynamics.* Experimental arithmetic, high-speed computations and mathematics, 1986. 2, 16

[J.92] Monaghan J. *Smoothed particle hydrodynamics.* Ann. Rev. Astron. Astrophys, 1992. 2

[LM10] Fedkiw R. Lentine M., Zheng W. *A novel algorithm for incompressible flow using only a coarse grid projection.* In Proc. SIGGRAPH, 2010. 2, 34

[MM03] Gross M. Müller M., Charypar D. *Particlebased fluid simulation for interactive applications.* Proceedings of the 2003 ACM SIGGRAPH, 2003. 2

[MM07] Duthaler S. Müller M., Schirm S. *Screen Space Meshes.* ACM SIGGRAPH Symposium on Computer Animation, 2007. 36

[R.08] Bridson R. *Fluid Simulation for Computer Graphics.* A K Peters, London, UK, 2008. 3, 14, 22, 23, 25, 36

[SD95] H. L. Schreyer Sulsky D., S.J. Zhou. *Application of a particlein-cell method to solid mechanics.* Comput. Phys. Commun, 1995. 3

[Sta99] Jos Stam. *Stable Fluids.* Proc. SIGGRAPH, 1999. 2, 3, 13

[Zha05] Hongkai Zhao. *A fast sweeping method for Eikonal equations.* Math. Comp, 2005. 18

[ZY05] Bridson R. Zhu Y. *Animating sand as a fluid.* In Proc. SIGGRAPH, 2005. 2, 3, 16, 18, 34