Real Time Multiple Fluid Simulation and Rendering on GPUs

Dunfan Lu

Somerville College

University of Oxford

Supervised by: Joe Pitt-Francis

3rd Year Project Report for

MCompSci

Trinity 2020

Abstract

Fluid simulation is an extremely common and important computational task. These simulations are often heavily expensive and require a large amount of CPU time, hence difficult to apply in real time applications. Fortunately, modern GPUs, equipped with massively parallel general purpose computing architectures, provide a solution to this problem. This project explores methods to perform fluid simulations on GPUs, and to realistically render the simulated fluids, both in real time.

This project focuses on three of the most widely used fluid simulation algorithm: FLIP (Fluid-Implicit-Particle), PBF (Position Based Fluids), and PCISPH (Predicative-Corrective Incompressible Smoothed Particle Hydrodynamics). The project studies how each algorithm can be parallelized, and creates efficient GPU implementations of these algorithms using NVIDIA's CUDA programming model. This project also extends the FLIP algorithm to support multiple fluid rendering, thereby capturing the diffusion phenomenon between different phases of fluids. Alongside the simulation, a real time liquid rendering scheme is implemented, which efficiently captures the refection and refraction effects, as well as the varying concentrations of colored fluid phases inside the liquid. These algorithms are integrated into a fully featured program, which accepts user-provided simulation parameters, then performs and renders the liquid simulation in a real time and visually plausible manner.

Contents

1 Introduction

1.1 Motivation

Fluids can be seen everywhere. The smoke coming out of chimney spreading in the wind, the milk in a cup mixing with the coffee, the calm flow of a river with tiny ripples under the rain, and the enormous waves of the ocean splashing onto the surface. Many of these these phenomenons have interesting and even stunning visual effects, thus, quite often, plausible images of these fluids need to be computationally generated for purposes such as cinematics and video gaming.

Due to the complexity underlying the behavior of fluids, accurate numerical simulation of fluids often require a huge amount of computational resources. In areas such as aeronautic engineering, where the importance of accuracy completely overrides that of efficiency, hours of CPU time can be spent on the simulation of a few seconds of fluid motion. Real time computer graphics applications, such as video games, usually do not require this level of accuracy, but instead asks the simulation to be computed in roughly the same amount of time as the physical process it represents. This efficiency requirement is met with the help of modern GPUs, which have massively parallel computing abilities. This project demonstrates this by implementing three simulation algorithms on GPUs, all of which can perform simulations in real time.

For computer graphics applications, fluid simulation isn't the only task. It is equally necessary for the software to display the results of simulation (i.e, the shape and motion of the fluid) to the user. This is especially important for interactive video games, where the user experience highly depends on the quality of the rendering. This project thus studies and implements the real time photorealistic rendering of liquids, the most frequently simulated and visualized type of fluid.

1.2 Related Work

The study of the behavior of fluids dates back to 18th century, when Leonhard Euler proposed a set of partial differential equations (PDEs), known as *Euler Equations*, that governs the behavior of an idealized incompressible and inviscid fluid. In the 19th century, these equations were extended by Claude-Louis Navier and George Gabriel Stokes into the famous *Navier-Stokes Equations*, which describe a much wider class of fluids that occur in the real world. These equations are explained in greater details in chapter 2, and they are exactly what most fluid simulation softwares, including the one implemented in this project, are trying to solve.

Somewhat unfortunately, the Euler and Navier-Stokes equations have extremely difficult mathematical properties, and general analytical solutions are yet to be found even today. As a result, softwares resort to numerical methods to approximate solutions. In computer graphics applications, there are two main families of numerical methods for solving the fluid equations: the grid-based methods and the particle-based methods. Each approach comes with its own benefits and drawbacks, but could both be implemented efficiently on GPUs to achieve real time simulation.

The grid-based methods relies on spatial discretizations of the fields (e.g the velocity field) that represents the fluids. The most widely used discretization method, known as the **MAC** (Marker and Cell) **grid**, was proposed by Harlow and Welch [3] in 1965. This scheme offers second order accuracy, and is used as a basis of most grid-based fluid simulation algorithms.

A significantly important step during a grid-based simulation is to move all the physical quantities stored in the grid (e.g concentration) according to the velocity field. This step, known as **advection**, essentially determines how the shape of the fluid evolves over time, thus is key to a high-fidelity simulation. A few popular advection algorithms include **MacCormack**[9] and **BFECC**[5], both of which have efficient GPU implementations[2][11]. This project chooses to implement the advection algorithm known as **FLIP** (Fluid Implicit Particle)[12], developed by Zhu and Bridson. This algorithm, interestingly enough, makes uses of particles to move quantities within the MAC grid. FLIP has various advantages over the purely grid-based algorithms, and is likely the most widely used advection method nowadays.

As an addition to the traditional single phase fluid simulation, Kang et el.[4] showed how to extend the grid-based algorithms to capture the diffusion between multiple fluid phases (e.g red ink diffusing in transparent water). This project imple-

ments a modified version of the proposed algorithm, where FLIP, rather than BFECC, is used to advect the concentration of different fluid phases.

Parallel to the grid-based approach is the family of particle-based algorithms. For computer graphics, the most commonly used class of particle-based algorithms is known as **SPH** (Smoothed Particle Hydrodynamics). Introduced to computer graphics in 2003 by Müller [8], the SPH method represent the fluid by a moving cloud of particles, which carry the physical quantities with them. This project chooses to implement two extensions to SPH: **PCISPH**(Predicative-Corrective Incompressible SPH) by Solenthalar[10] and **PBF**(Position Based Fluids) by Macklin and Müller[7]. These extended algorithms improve upon the plain SPH in that they enforce the incompressibility constraint of fluids, which are important for visual fidelity.

Given a well performing simulation, either grid-based or particle-based, it remains a nontrivial task to visualize the fluid. This project follows the proposal by Zhu and Bridson [12], who showed how a particle representation of a fluid can be used to compute a signed distance field, which represents the distance to the fluid surface of each point in the 3D space. An algorithm known as Marching Cubes [6], proposed by Lorensen, can then use this field to reconstruct the surface of the fluid into a triangle mesh representation, which is suitable for rendering.

1.3 Project Outline

This project focuses on investigating and producing highly performant GPU implementations of the most widely used fluid simulation and rendering algorithms. An extended version of the FLIP algorithm, which supports multiple fluid simulation and diffusions between fluids of different colors, is studied and implemented, with the details elaborated in chapter 4. Similarly, GPU versions of the PCISPH and PBF algorithm are also created, as described in chapter 5.

To visualize the simulations, the project implements a fast surface reconstruction algorithm, which transforms a particle cloud representation of fluids into a renderable triangle mesh. A real time renderer is implemented to render the mesh while capturing all the reflection and refraction phenomenons that occur in the real world. Furthermore, the renderer takes into account the different levels of attenuation of light caused by fluids of different colors, thereby also realistically rendering the liquid diffusion effects. The details of the renderer are given in chapter 6.

These implementations are based from their origin descriptions in the papers, but many additional considerations and optimizations were taken to enable efficient

parallelization. Specifically, the project utilizes NVIDIA's general purpose GPU programming interface known as CUDA, and tailors the implementation code to exploit the full potential of CUDA GPUs. The results are showcased by a fully featured program, which allows the user to easily configure the starting state of a simulation. These include the shapes and sizes of the fluid before the simulation starts, as well as the initial color and transparency of each fluid volume. The program can then carry out the simulation and render realistic results to the user in real time.

2 Physics of Fluids

The mechanics of fluids are governed by the partial differential equations (PDEs) known as the *Incompressible Navier-Stokes Equations*, or in case of inviscid fluids, the *Euler Equations*. This chapter explains the meaning and intuition behind these equations, which are key to designing and implementing numerical simulation algorithms.

2.1 Vector Calculus

The fluid equations are commonly written in the language of vector calculus. A brief introduction of the main concepts and operators involved is given in this chapter.

Scaler Field

A *scalar field* on $\mathbb{R}^3$ is a mapping $\phi : \mathbb{R}^3 \to \mathbb{R}$ from 3D cartesian coordinates to scalar values. Example scalar fields include fluid density, or pressure, where a scalar value can be sampled in each point of the 3D space.

Vector Field

A *vector field* on $\mathbb{R}^3$ is a mapping $\phi : \mathbb{R}^3 \to \mathbb{R}^3$ from 3D cartesian coordinates to 3D vectors. A very commonly used vector field is the velocity field $\mathbf{u}$, which describes the direction and speed of the fluid's movement at each point in the 3D space

The grad

Given a scalar field $\phi : \mathbb{R}^3 \to \mathbb{R}$, the *gradient* or *grad* of the field is a vector field written as $\nabla\phi$, and it is defined by:

$$\nabla\phi = \begin{pmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \\ \dfrac{\partial\phi}{\partial z} \end{pmatrix}$$

The grad of a scalar quantity ϕ represents the rate of change of ϕ across each dimension. Moreover, $\nabla\phi$ computes the direction of movement which causes the greatest increase of ϕ.

The div

Given a vector field $\mathbf{u} : \mathbb{R}^3 \to \mathbb{R}^3$, the *divergence* or *div* of the field is a scalar field written as $\nabla \cdot \mathbf{u}$, and it is defined by:

$$\nabla \cdot \mathbf{u} = \nabla \cdot \begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix} = \frac{\partial \mathbf{u}_x}{\partial x} + \frac{\partial \mathbf{u}_y}{\partial y} + \frac{\partial \mathbf{u}_z}{\partial z}$$

If $\mathbf{u}$ is the velocity field, then the scalar field $\nabla \cdot \mathbf{u}$ represents the speed at which the fluid is expanding or shrinking at each 3D location. Thus, a velocity field that satisfies $\nabla \cdot \mathbf{u} = 0$ would keep the fluid in constant volume, which is how most fluids behave in the real world.

The curl

Given a vector field $\mathbf{u} : \mathbb{R}^3 \to \mathbb{R}^3$, the *curl* of the field is a scalar field written as $\nabla \times \mathbf{u}$, and it is defined by:

$$\nabla \times \mathbf{u} = \nabla \times \begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix} = \begin{pmatrix} \frac{\partial \mathbf{u}_z}{\partial y} - \frac{\partial \mathbf{u}_y}{\partial z} \\ \frac{\partial \mathbf{u}_x}{\partial z} - \frac{\partial \mathbf{u}_z}{\partial x} \\ \frac{\partial \mathbf{u}_y}{\partial x} - \frac{\partial \mathbf{u}_x}{\partial y} \end{pmatrix}$$

Informally, the curl of the velocity field is a measure of the local rotation of the fluid. Though not directly used in the equations and algorithms presented in this project, it is at the heart of a different class of algorithms, called the vortex methods[1].

The Laplacian

The *Laplacian* operator, written $\nabla \cdot \nabla$, is defined to be the divergence of the gradient. For scalar field ϕ, it can be computed that:

$$\nabla \cdot \nabla\phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2}$$

The Laplacian describes the difference between the average value of ϕ in the neighborhood of a certain point and the value of ϕ at that point. As defined, this operator

takes a scalar field and returns a scalar field. However, The Laplacian is also often extended to be applied to vector fields, where

$$\nabla \cdot \nabla \mathbf{u} = \begin{pmatrix} \nabla \cdot \nabla \mathbf{u}_x \\ \nabla \cdot \nabla \mathbf{u}_y \\ \nabla \cdot \nabla \mathbf{u}_z \end{pmatrix}$$

2.2 The Eulerian and Lagrangian Viewpoints

For any physical quantity that represents some property of a fluid, the field of that quantity, either scalar or vector, could be constantly evolving as time passes. There are two different approaches to tracking this rate of change: the Eulerian viewpoint and the Lagrangian viewpoint.

The Eulerian viewpoint considers the time derivative of quantities at fixed locations in the 3D space. For a scalar field ϕ which varies through time, its *Eulerian derivative* is simply $\dfrac{\partial \phi}{\partial t}$. To be more precise, the Eulerian derivative $\dfrac{\partial \phi}{\partial t}$, evaluated at point $\mathbf{x}$, is the rate of change of ϕ of the fluid at the fixed position $\mathbf{x}$, despite the fact that the fluid could be in motion. This has the immediate consequence that the concept of Eulerian derivative fails to capture the fact that physical quantities are carried around (i.e advected) by the fluid.

The Lagrangian viewpoint, on the other hand, tracks the rates of changes of quantities as it moves along the velocity field $\mathbf{u}$. In this approach, for a scalar field ϕ, its derivative with respect to time is written as $\dfrac{D\phi}{Dt}$, and defined to be

$$\frac{D\phi}{Dt} = \frac{\partial \phi}{\partial t} + \nabla \phi \cdot \mathbf{u}$$

This derivative, known as the *Lagrangian derivative* or *material derivative*, can be justified by treating the fluid as a collection of infinitesimal particles, each carrying some quantities and moving along the velocity field. At time t, for each particle p with position $\mathbf{x}$, the quantity of ϕ it carries is $\phi_p = \phi(t, \mathbf{x}(p))$. The derivative with respect to t of this term computes the rate of change of ϕ_p:

$$\begin{aligned} \frac{d}{dt}\phi_p &= \frac{d}{dt}\phi(t, \mathbf{x}(t)) \\ &= \frac{\partial \phi}{\partial t} + \nabla \phi \cdot \frac{d\mathbf{x}}{dt} \\ &= \frac{\partial \phi}{\partial t} + \nabla \phi \cdot \mathbf{u} \\ &= \frac{D\phi}{Dt} \end{aligned}$$

Which is precisely the Lagrangian derivative.

When formalizing the Euler and Navier-Stokes equations, the Lagrangian derivative $\frac{D}{Dt}$ will be automatically extended to be applied to vector fields, where each component of the vector field is differentiated separately. This allows the term $\frac{D\mathbf{u}}{Dt}$ to be written, representing the acceleration of the infinitesimal fluid particles:

$$\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + \begin{pmatrix} \nabla \mathbf{u}_x \cdot \mathbf{u} \\ \nabla \mathbf{u}_y \cdot \mathbf{u} \\ \nabla \mathbf{u}_z \cdot \mathbf{u} \end{pmatrix} \tag{2.1}$$

As a flashforward to chapter 4 and 5, the grid-based methods mainly employ the Eulerian viewpoint, storing quantities on a fixed grid, and using an explicit computational step called *advection* to move around the quantities. In contrast, the particle-based methods always use the Lagrangian viewpoint, with quantities being recorded solely on particles.

2.3 The Euler and Navier-Stokes Equations

Using the previously defined notations, the Euler equations, which governs the motion of an incompressible and inviscid fluid , can be written as

$$\begin{cases} \dfrac{D\mathbf{u}}{Dt} = -\dfrac{\nabla p}{\rho} + \mathbf{g} \\ \nabla \cdot \mathbf{u} = 0 \end{cases} \qquad \text{(Euler Equations)}$$

where $\mathbf{u}$ is the velocity field, p is pressure, ρ is the fluid's density, and $\mathbf{g}$ the acceleration caused by an external force field (e.g gravity).

A generalized version of these equations is the famous incompressible Navier-Stokes equations, in which a term that describes viscosity is added:

$$\begin{cases} \dfrac{D\mathbf{u}}{Dt} = -\dfrac{\nabla p}{\rho} + \mathbf{g} + \nu \nabla \cdot \nabla \mathbf{u} \\ \nabla \cdot \mathbf{u} = 0 \end{cases} \qquad \text{(Navier-Stokes Equations)}$$

where ν is the kinematic viscosity coefficient.

As described in the last section, the quantity $(\nabla \cdot \mathbf{u})$ represents the rate at which the fluid is expanding or shrinking. Fluids in the real world usually remains in constant volume, unless in extreme conditions. This motivates the equation $\nabla \cdot \mathbf{u} = 0$, included in both Euler and Navier-Stokes.

Besides the incompressibility condition, both Euler and Navier-Stokes include another equation known as the momentum equation (which is in fact a set of equations,

because the quantities are vectors). The momentum equation essentially specifies Newton's 2nd law: $\mathbf{a} = \dfrac{\mathbf{F}}{m}$, i.e the acceleration is the force divided by the mass.

As previously explained, the quantity $\dfrac{D\mathbf{u}}{Dt}$ represents the acceleration of the infinitesimal fluid particles. Thus, to explain the momentum equations, it remains to demonstrate that the right hand side correctly computes the force divided by the mass. Let the mass of the infinitesimal particle be m, and let the force be separated into the internal forces within the fluid F_{in} and the external forces F_{ext}:

$$\frac{D\mathbf{u}}{Dt} = \frac{F_{in} + F_{ext}}{m}$$

With $\mathbf{g}$ representing the acceleration caused by an external force field (e.g gravity), this can be rewritten as

$$\frac{D\mathbf{u}}{Dt} = \frac{F_{in}}{m} + \mathbf{g}$$

The internal forces within a fluid is caused by an imbalance in pressure. Specifically, if one side of a infinitesimal particle has a greater pressure than the opposite side, then the particle will be pushed towards the low pressure region. This justifies why the pressure forces are in the direction of $-\nabla p$, which computes the direction of fastest decrease of pressure. From a dimensional analysis point of view, the unit of p is $\dfrac{force}{length^2}$, thus the unit of ∇p is $\dfrac{force}{length^3}$. This indicates that to obtain the pressure force, it's necessary to multiply by the volume V of the infinitesimal particle, which produces

$$\frac{D\mathbf{u}}{Dt} = -\frac{V\nabla p}{m} + \mathbf{g}$$

Using $\rho = \dfrac{m}{V}$, this becomes:

$$\frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho} + \mathbf{g}$$

Which is Euler's momentum equation. It is important to note that the justifications of the fluid equations given in this section is far from a rigorous mathematical derivation, which would not fit into this report due to its complexity.

The Navier-Stokes momentum equation extends the Euler momentum equation by considering viscosity. In a viscous fluid, the velocity of a particle tends to diffuse into its surrounding particles, causing the velocity in the neighborhood to converge into its average. The difference between the average of $\mathbf{u}$ in the neighborhood and the value of $\mathbf{u}$ of the particle is captured by the Laplacian of the velocity: $\nabla \cdot \nabla \mathbf{u}$, thus adding a positive multiple of this quantity creates a viscous effect:

$$\frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho} + \mathbf{g} + \nu \nabla \cdot \nabla \mathbf{u}$$

where ν is a constant property, known as the kinematic viscosity of the fluid. For water, which is a rather inviscid fluid, this quantity is almost negligible. When simulating water, considering the effects of viscosity requires considerable extra computation, while bringing little improvements to the visual fidelity. As a result, this project chooses to only solve the Euler equations during simulation.

2.4 Boundary Conditions

For a fluid region that is not the entirety of $\mathbb{R}^3$, boundary conditions must be specified, which defines the behavior of the fluid in the physical boundary of the fluid region.

When simulating liquids, there are two types of boundary conditions: the solid boundaries and the free surface boundaries. At a solid boundary, the condition is

$$\mathbf{u} \cdot \mathbf{n} = 0$$

where $\mathbf{n}$ is the normal of the solid surface. This condition ensures that the fluid cannot flow into a solid.

The second type of boundary is the free surface boundary, which is the boundary between the liquids and some region of space that is not occupied by anything. In this case, that region of space will not exert any force. and therefore pressure, to the fluid, which motivates the condition

$$p = 0$$

This free surface condition can be also applied to the boundary between liquid and air, which is because air is significantly lighter than liquid, and hence does not influence the motion of the liquid.

The liquid simulated in this project is contained within a cubic box. Moreover,it doesn't fill the box entirely and thus has a free surface. Consequently, both boundary condition will be applied during the numerical simulation.

2.5 The Diffusion Equation

Finally, this section introduces an equation that governs the concentration changes in a mixture of more than one miscible fluids. Let α be the scalar field representing the concentration of a certain fluid, the diffusion equation specifies:

$$\frac{\partial \alpha}{\partial t} = D\nabla \cdot \nabla \alpha \qquad \text{(Diffusion Equation)}$$

Where D is the diffusion coefficient. This is again justified by that fact that the Laplacian $D\nabla \cdot \nabla\alpha$ captures the difference between the average value of α in the neighborhood of a 3D location and the value of α at that location.

3 GPU Programming

To achieve maximum performance, this project chooses to implement the fluid simulation software on GPUs. Originally built for graphics applications, GPUs were designed to handle a massive amount of geometries and pixels in parallel, because the computation for different objects and pixels are largely independent. The ability to do massively parallel computation motivated GPGPU (General Purpose GPU) programming models to arise, which became significantly useful for scientific computing purposes. The implementation code in this project is written using the CUDA programming model, developed by the NVIDIA Corporation.

3.1 The CUDA Programming Model

The parallel computation ability of GPUs comes from its intrinsically parallel architecture.

3.2 The OpenGL Render Pipeline

4 Grid-Based Simulations

This chapter introduces a grid-based multiphase fluid simulation scheme and its CUDA implementation in this project. This scheme has three key components: a **MAC** (Marker and Cell) grid for discretizing the Euler equations, a **FLIP** (Fluid Implicit Particle) algorithm for advection, and a Jacobi linear solver for solving the diffusion equation and the Poisson pressure equation (which ensures incompressibility).

4.1 Operator Splitting

A common way for numerically solving differential equations is the *splitting* approach. As a simple example, consider the simple differential equation:

$$\frac{dx}{dt} = f(x) + g(x) \quad \text{With boundary condition } x(0) = x_0$$

To numerically solve this, decide on some small time step $\triangle t$, and let $x_{[n]}$ be the value of x at the nth time step. The goal is to find $x_{[n]}$ for increasing larger n. To do this, start with $x_{[0]} = x_0$ and consider the two differential equations:

$$\frac{dx}{dt} = f(x)$$
$$\frac{dx}{dt} = g(x)$$

Suppose there exists some good solutions (either analytical or numerical) for these two equations, then these solutions can be used to find a good solution for the original equation. Specifically, suppose $F_{f_0}(t)$ is a solution of $\frac{dx}{dt} = f(x)$ with boundary condition $x(0) = f_0$, and $G_{g_0}(t)$ is a solution of $\frac{dx}{dt} = g(x)$ with boundary condition $x(0) = g_0$, then, the original equation can be solved as

$$\widetilde{x} = F_{x_{[n]}}(\triangle t)$$
$$x_{[n+1]} = G_{\widetilde{x}}(\triangle t)$$

In essence, this approach splits the equation into a few more easily solved differential equations, and accumulates the solution of each over a small time step.

This same splitting approach can be applied to the Euler equations. To do so, the Euler momentum equation is first written in a form where the material derivative is expanded using equation (2.1):

$$\frac{\partial \mathbf{u}}{\partial t} = -\begin{pmatrix} \nabla \mathbf{u}_x \cdot \mathbf{u} \\ \nabla \mathbf{u}_y \cdot \mathbf{u} \\ \nabla \mathbf{u}_z \cdot \mathbf{u} \end{pmatrix} + \mathbf{g} - \frac{\nabla p}{\rho}$$

This then allows the equation, and therefore the simulation algorithm, to be split into three parts:

1. The equation
$$\frac{\partial \mathbf{u}}{\partial t} = -\begin{pmatrix} \nabla \mathbf{u}_x \cdot \mathbf{u} \\ \nabla \mathbf{u}_y \cdot \mathbf{u} \\ \nabla \mathbf{u}_z \cdot \mathbf{u} \end{pmatrix}$$
Again using equation (2.1), this can be rewritten back into the material derivative form:
$$\frac{D\mathbf{u}}{Dt} = 0$$
Intuitively, solving this equation means to move the fluid according to its velocity field, in a way such that the velocity of each infinitesimal fluid partial remains unchanged. This is the step known as *advection*.

2. The equation
$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g}$$
Solving this equation is the process of exerting external forces (e.g gravity) on the fluid, straightforwardly achievable by adding $\triangle t\mathbf{g}$ to velocity field output by the advection step.

3. The equation
$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{\nabla p}{\rho}$$
Importantly, since this is the last step of the splitting, it is essential to make sure that the results of solving this equation satisfies the incompressibility condition $\nabla \cdot \mathbf{u} = 0$. This amounts to finding a pressure field p such that, subtracting by $\triangle t\frac{\nabla p}{\rho}$ makes the velocity have zero divergence. This step enforces the incompressibility of the fluid.

4.2 Discretization

Throughout the simulation, a discretized representation of the velocity is needed. A obvious choice is to maintain a 2D grid, with each grid cell storing a vector quantity that is the velocity sampled at the center of the grid cell. This projects implements a slightly more sophisticated method, known as the **MAC**(Marker and Cell) grid, which offers more convenience and higher accuracy when computing the divergence $\nabla \cdot \mathbf{u}$ during the incompressibility step.

Instead of storing the value of $\mathbf{u} = (\mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z)$ sampled at the cell center, an MAC grid stores different components of $\mathbf{u}$ sampled at different locations. Specifically, the grid cell at position (x, y, z) stores the value of $\mathbf{u}_x$ sampled at the center of its left face, it stores the value of $\mathbf{u}_y$ sampled at its lower face, and it stores the value of $\mathbf{u}_z$ sampled at its back face, as illustrated in this figure:

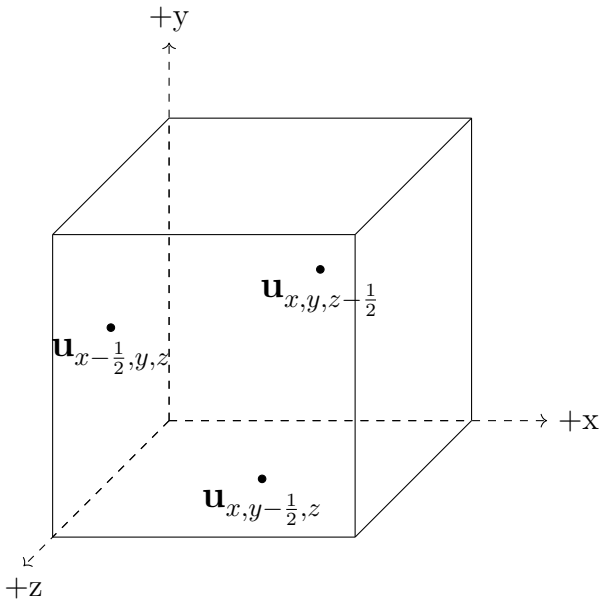
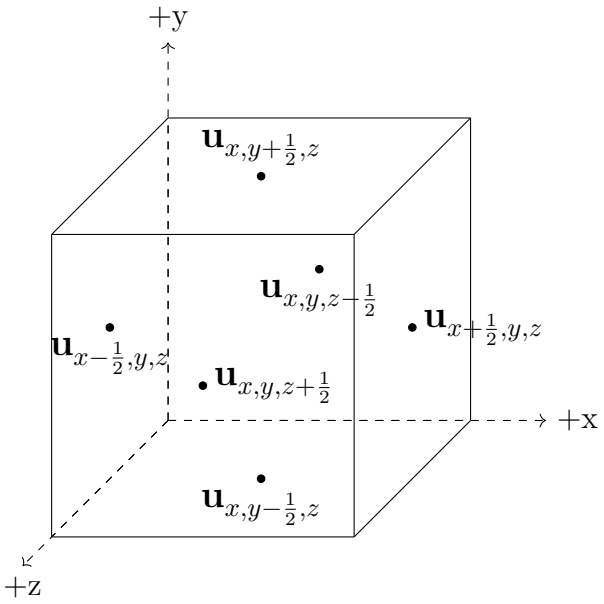


Figure 4.1: a 2D MAC grid cell and the velocity data it stores

The quantities $\mathbf{u}_{x,y,z-\frac{1}{2}}$, $\mathbf{u}_{x,y-\frac{1}{2},z}$, $\mathbf{u}_{x-\frac{1}{2},y,z}$ are all scalars, representing the velocity pointing at the x, y, and z direction, respectively. Furthermore, notice that the values of $\mathbf{u}_{x+\frac{1}{2},y,z}$, $\mathbf{u}_{x,y+\frac{1}{2},z}$, and $\mathbf{u}_{x,y,z+\frac{1}{2}}$, which are respectively sampled at the right, upper, and front faces, will also be available. This is because $\mathbf{u}_{x+\frac{1}{2},y,z} = \mathbf{u}_{x+1-\frac{1}{2},y,z}$, that is, the value of $\mathbf{u}_x$ sampled at the right face of the cell is exactly the value of $\mathbf{u}_x$ sampled at the neighboring cell to the right. The same can be applied for the upper and front faces. As a result, there are 6 velocity values associated with each grid cell:



Using these quantities, a finite difference approximation of the value of the divergence of the velocity, $\nabla \cdot \mathbf{u}$, can be easily computed:

$$\begin{aligned}
\nabla \cdot \mathbf{u} &= \frac{\partial \mathbf{u}_x}{\partial x} + \frac{\partial \mathbf{u}_y}{\partial y} + \frac{\partial \mathbf{u}_z}{\partial z} \\
&\approx \frac{\triangle \mathbf{u}_x}{\triangle x} + \frac{\triangle \mathbf{u}_y}{\triangle y} + \frac{\triangle \mathbf{u}_z}{\triangle z} \\
&= \frac{\mathbf{u}_{x+\frac{1}{2},y,z} - \mathbf{u}_{x-\frac{1}{2},y,z}}{\triangle x} + \frac{\mathbf{u}_{x,y+\frac{1}{2},z} - \mathbf{u}_{x,y-\frac{1}{2},z}}{\triangle x} + \frac{\mathbf{u}_{x,y,z+\frac{1}{2}} - \mathbf{u}_{x,y,z-\frac{1}{2}}}{\triangle x}
\end{aligned} \tag{4.1}$$

In the MAC grid, only the velocity field is stored in this pattern. Other scaler quantities, such as pressure, are all sampled at the center of each grid cell. This arrangement offers a convenient discretization of the equation $\frac{\partial \mathbf{u}}{\partial t} = -\frac{\nabla p}{\rho}$, used in step 3 of the simulation.

5 Particle-Based Simulations

6 Rendering

A Sample Title

B Sample Title

Bibliography

[1] Alexis Angelidis and Fabrice Neyret. Simulation of smoke based on vortex filament primitives. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 87–96, 2005.

[2] Nuttapong Chentanez and Matthias Müller. Real-time eulerian water simulation using a restricted tall cell grid. In *ACM Siggraph 2011 Papers*, pages 1–10. 2011.

[3] Francis H Harlow and J Eddie Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[4] Nahyup Kang, Jinho Park, Junyong Noh, and Sung Yong Shin. A hybrid approach to multiple fluid simulation using volume fractions. In *Computer Graphics Forum*, volume 29, pages 685–694. Wiley Online Library, 2010.

[5] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jaroslaw R Rossignac. Flowfixer: Using bfecc for fluid simulation. Technical report, Georgia Institute of Technology, 2005.

[6] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[7] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

[8] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[9] Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2-3):350–371, 2008.

[10] Barbara Solenthaler and Renato Pajarola. Predictive-corrective incompressible sph. In *ACM SIGGRAPH 2009 papers*, pages 1–6. 2009.

[11] Shibiao Xu, Xing Mei, Weiming Dong, Zhiyi Zhang, and Xiaopeng Zhang. Interactive visual simulation of dynamic ink diffusion effects. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 109–116, 2011.

[12] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)*, 24(3):965–972, 2005.