Playing Poseidon: A Lattice Boltzmann Approach to Simulating Generalised Newtonian Fluids

Thesis submitted in partial fulfillment
of the requirements for the degree of

Dual Degree (BTech + MS by Research)
in
Computer Science

by

Nitish Tripathi
200802027
nitish.tripathi@research.iiit.ac.in

Center for Visual Information Technology
International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2016

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Playing Poseidon: A Lattice Boltzmann Approach to Simulating Fluids" by Nitish Tripathi, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Dr. P. J. Narayanan

You could not step twice into the same rivers;
for other waters are ever flowing on to you.

Heraclitus

Acknowledgments

Gurur Brahma Gurur Vishnu Gurur Devo Maheshwara

I would like to thank my parents whose guidance has subtly ensured that I do not stray. Dr. P. J. Narayanan provided the same guidance during my academic journey in IIIT. A thank you would be an insignificant acknowledgement of the patience he showed while seeing me err time and again during research.

Mention must also be made of Rishi Gupta and Parikshit Sakurikar. The former for betting I couldn't put the word "Fractal" in my thesis (Ha!) and the latter for ensuring that no stone was left unturned while ensuring I miss deadlines.

Last but not the least, I would like to thank the gang who put up with me for the better part of my stay on campus. This is definitely not a forced acknowledgement and they definitely have not blackmailed me into putting their names in here. They also have definitely not told me to put their names in else they will not let me submit my thesis. For want of a better word, they are my friends. Foremost among these non-extortioners and non-blackmailers are Naveen Sinha, Jay Guru Panda and Antarip Halder.

Contents

List of Figures

List of Tables

Chapter 1

A brief history of fluid simulation

Imitating the behaviour and characteristics of fluids with the help of a computer is called fluid simulation. Real world fluids are fickle. They are subtle and gentle at times; at times they are ravenous and tumultuous. Needless to say, complex equations are behind even the tiniest of ripple, so much so that often fluid mechanics has been described as "*the physicists nightmare*". Yet there are few, if any, substances which are so beautiful and graceful in motion to observe. To an ardent student of hydrodynamics, *everything in the discernible world is fluid*. Solids may just be classified as fluids which flow extremely slowly! Given time, every substance has the tendency to flow under the influence of an external force.

The history of fluid simulations, thus, rightly, begins with the formulation of the Navier Stokes' equations. These were a set of partial differential equations originally developed in the 1840s on the basis of conservation laws and first order approximations. What followed was the conventional study of fluid flows for more than a century. Arriving at computational models to solve fluid equations has been a subject of research since the early 1950s. Finding solutions to the partial differentials of Navier Stokes' equations using discrete algorithms was area of focus. Many modern day techniques, of which some will be skimmed through in the succeeding chapters, came up during that time. Staggered marker-and-cell (MAC grid structure), Particle in Cell (PIC method) etc. are two of those.

However, most of the models and techniques developed by the CFD community then was complex and unscalable for *visual effects oriented computer graphics*. In the succeeding years fluid effects was generated using non-physics based methods, such as using hand drawn animation (key frame animation) or displacement mapping. The seminal work of Foster *et al.* [1] proved to be the adrenalin shot of Computer Graphics. This was the begining of Eulerian fluid simulation method. Stam *et al.* [2] was also one of the early contributors to the field. Since then, the fluid simulation engines have come a long way with engines now reproducing complicated behaviour with increasing ease.

1.1 A brief glimpse on Fluid Simulation Development

The development of fluid simulations from the time of Stam *et al.* [2] has traditionally been in two concurrent streams, viz., Eulerian and Lagrangian Simulations. Eulerian Method involves modelling the fluid as a collection of scalar fields (density, pressure etc.) and vector fields (velocity etc.). Each field is calculated using Navier Stokes' equations and fluid is visualised as crossing the volume at fixed grid points, where the value of each field is known. Lagrangian simulations on the other hand take a more intuitive approach. They treat fluid particles as carriers of the field values in accordance with the Navier Stokes' equations. The dependence of both the methods on Navier Stokes' makes them essentially top down simulations methods - these methods look at what the perceptible fluid properties are without concerning themselves about the kind of particular interactions which give rise to the said properties.

Lattice Boltzmann Method developed around the same time but did not come into widespread usage until much later. Unlike the conventional methods, it is a statistical method based on Kinetic Theory. It treats fluids as a collection of logical mesoscopic particles. These are constrained to move in a discrete set of directions across a cartesian grid. They follow continuous alternating iterations of colliding at each grid centre and redistribution around it, and, progreesing to the neighbouring centre. It was shown that for a particular kind of collisions these particular interactions give rise to Navier Stokes' properties at the macro level. However, we can tweak certain parameters during the development so that quantities, taken to essentially be constants in the final Navier Stokes' equations, can be varied to simulate fluids outside their scope. As we will see in the succeeding chapters, such fluids are in abundance around us and are called non-Newtonian fluids. The Navier Stokes' equation, as will also be seen in succeeding chapters, are essentially Newton's second law of motion. It is therefore unfit to deal with fluids which are non-Newtonian in nature and requires regular tweaking to model them.

1.2 Contributions

In this work, we combine physical models of non-Newtonian fluids with Lattice Boltzmann Method. We give the CPU implementation, showing how easy it is to understand and code. We show how the method, inspite of its ease of implementation, doesn't compromise on physical realism or accuracy. We also give a model for GPU implementation for increased efficiency and interactive frame rates.

1.3 Layout

In this chhapter we have given a brief account of our motivation, a succinct history of fluid simulation methodologies while introducing Lattice Boltzmann Method. We have also mentioned the contrinutions of this work.

Chapter 2 gives a detailed account of Newtonian and non-Newtonian fluids as well as the mathematical models for the same. Chapter 3 gives the basic physics behind Navier Stokes' equations and the Lattice Boltzmann Method. It also shows the convergence of LBM to Navier Stokes' behavior at macro levels. Chapter 4 talks of the implementation of LBM on a computer - the development of algorithm - for *generalised Newtonian fluids*. Chapter 5 gives details of the GPU implementation, while Chapter 6 lays down the comparative results obtained from our imlpementation and establishes its physical and mathematical accuracy. Finally, Chapter 7 concludes the work and lays down the poosible areas for future work.

Chapter 2

An Introduction to non-Newtonian Fluids

The development of *fluid dynamical theory* can be divided into three phases. Each phase is the logical successor of the former and the progression of each has brought more maturity to the field, a process which is still continuing. The first phase dealt with ideal fluids, characterized by the non-viscous and incompressible nature. Considering laminar flow, the shear between adjoining layers doesn't give rise to any contact forces (friction).

The second phase developed the mathematics behind Newtonian fluids where the flow domain was split into two, the boundary layer and non-boundary layer. The modeling is done by treating the fluid in the latter as ideal whereas fluid friction is taken into account in the former. Thus this resulted in fine tuning the theory to cover the simplest kind of *real fluids*. However there are a lot of fluids found in the real word which do not conform to even the Newtonian approach to modeling fluids. These are called, non-Newtonian fluids. Examples include biological fluids such as blood, mucus, sinovial fluid, semen etc, multiphase mixtures such as slurries, emulsions, food products such as jams, jellies soap solutions and many more. In fact, Newtonian fluids actually form a minuscle minority in the kinds of fluids found around us making them more of a departure than the rule. We refer the reader to [3] for a comprehensive coverage of the succeeding sections.

2.1 Newtonian Flow

To understand non-Newtonian flow behaviour it is imperative that we understand Newtonian behavior. Consider steady state flow for a fluid between two parallel plates (Fig. 2.1), when a constant shearing force $\mathbf{F}$ is applied to one plate (let us assign it to the top without loss of generality). For a Newtonian flow, the relation between the resultant shear stress τ and shear strain is,

$$\frac{\mathbf{F}}{A} \;=\; \tau_{yx} \;=\; \mu\left(-\frac{dV_x}{dy}\right) \;=\; \mu\dot{\gamma}_{yx} \tag{2.1}$$

where, A is the area of crossection of the plates, $\mathbf{V}$ is the velocity of the fluid. The minus sign implies a resistive force. μ is the Newtonian viscosity of the fluid.

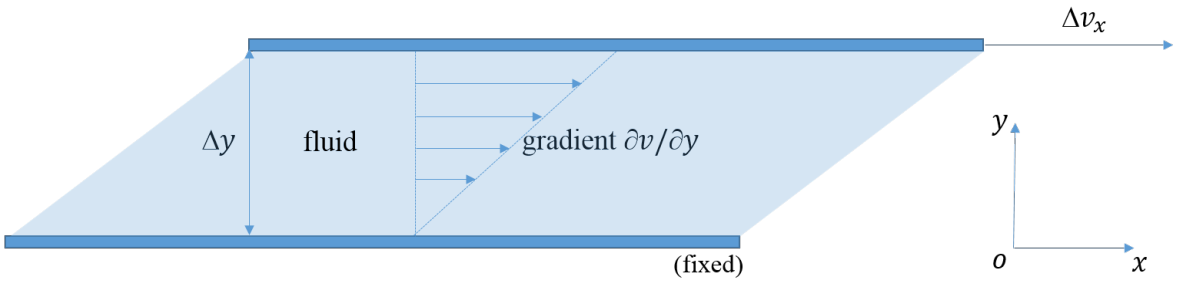

Figure 2.1: Newtonian flow between parallel plates

The graph of shear stress versus rate of shear is known as the flow curve for a fluid. It is evident from the relation that the curve for a Newtonian fluid shall be of the form $y = mx$, i.e., a straight line passing through origin with its slope being the viscosity (Fig. 2.2).

The relation provided is only for a two dimensional flow when the velocity is along the x-direction and varies along the y-direction. This is known as simple shear flow. In a more general 3D situation there would be other relations as well,

$$\tau_{yx} = -\mu\left(\frac{\partial V_x}{\partial y} + \frac{\partial V_y}{\partial x}\right) \tag{2.2}$$

$$\tau_{yy} = -2\mu\frac{\partial V_x}{\partial y} + \frac{2}{3}\mu\left(\nabla.\mathbf{V}\right) \tag{2.3}$$

$$\tau_{yz} = -\mu\left(\frac{\partial V_z}{\partial y} + \frac{\partial V_y}{\partial z}\right) \tag{2.4}$$

Similar expressions for x and z directions give rise to a 3×3 stress tensor. The normal stresses can be viewed as a combined result of fluid motion and pressure p.

$$P_{xx} = -p + \tau_{xx} \tag{2.5}$$

$$P_{yy} = -p + \tau_{yy} \tag{2.6}$$

$$P_{zz} = -p + \tau_{zz} \tag{2.7}$$

By definition, isometric pressure is given by,

$$p = -\frac{1}{3}\left(P_{xx} + P_{yy} + P_{zz}\right). \tag{2.8}$$

Thus we obtain,

$$\tau_{xx} + \tau_{yy} + \tau_{zz} = 0 \tag{2.9}$$

It is also a fact that deviatoric normal stress in a Newtonian flow in simple shear are identically zero.. That is,

$$\tau_{xx} = \tau_{yy} = \tau_{zz} = 0 \tag{2.10}$$

The previous relation, along with constant slope of the flow curve passing through origin completely define a Newtonian fluid.

2.2 Non-Newtonian Fluids

The best indicators of non-Newtonian behavior are the flow curves. If they are non-linear or linear but not passing through the origin then the corresponding fluids are non-Newtonian. The non-Newtonian behavior is classified into three categories,

1. Substances for which the rate of shear is dependent only on the current value of the shear stress or vice-versa; this class of materials is variously known as purely viscous, time-independent, or generalized Newtonian fluids (GNF).

2. More complex materials for which the relation between the shear stress and the shear rate also depends upon the duration of shearing, the previous kinematic history, etc; these are known as time-dependent systems.

3. Materials exhibiting combined characteristics of both an elastic solid and a viscous fluid and showing partial elastic and recoil recovery after deformation, known as visco-elastic fluids.

The flow curves of various kinds of non-Newtonian substances is provided in the Fig. 2.2.

It is not uncommon for materials to exhibit a combination of these characteristics or behave as different materials depending on the environmental conditions. However, the characteristic flow curve even for a fluid showing only pseudoplastic or only dilatant behavior may not correspond to a strictly monotonic decrease or increase in viscosity. An example of the relationship between viscosity of a non-Newtonian fluid the rate of shear applied to it is shown in Fig. 2.3.

2.3 Generalized Newtonian Flow

The behaviour of Generalized Newtonian Fluids (GNF) is independent of the time duration of the application of the shear stress. In other words, there is a one-to-one functional dependence of the rate of shear on the shear stress, that is,

$$\tau_{yx} \; = = f\left(\dot{\gamma}_{yx}\right) \tag{2.11}$$

GNF can fall in either of the following three categories,

1. Shear-thinning or pseudoplastic,

2. Shear-thickening or dilatant or,

3. Newtonian.

Psudoplastic fluids are those for which the viscosity decreases with increasing shear rate. Dilatant fluids are the opposite, with the viscosity increasing with increasing shear rate.

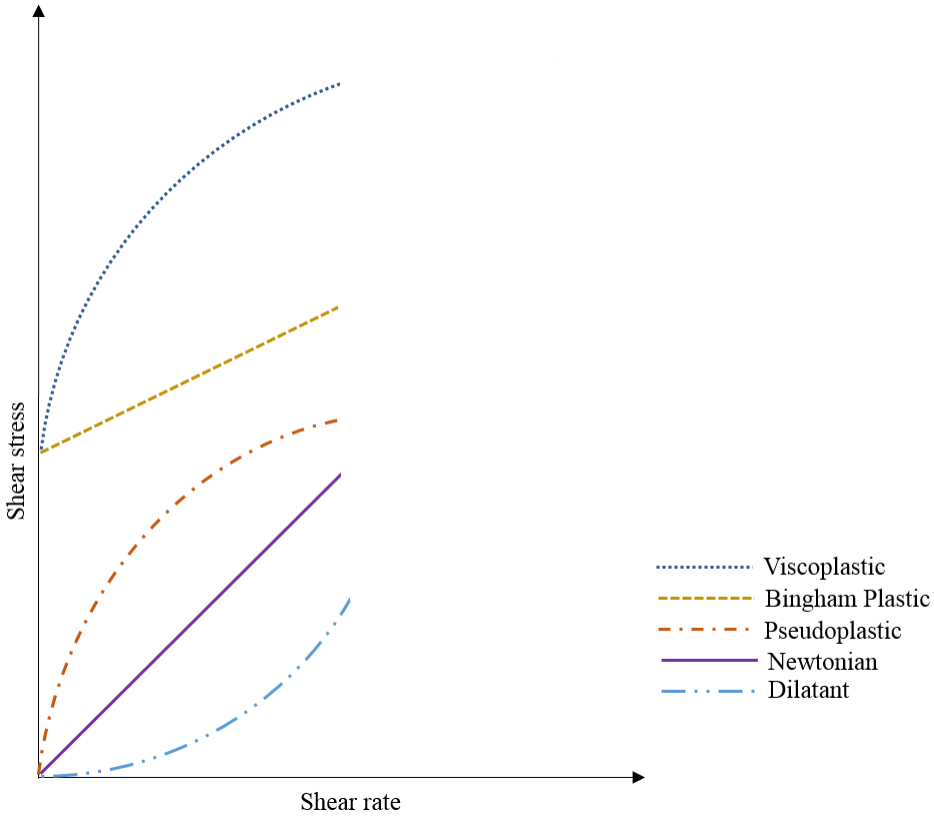

Figure 2.2: Flow Curve for different types of fluid

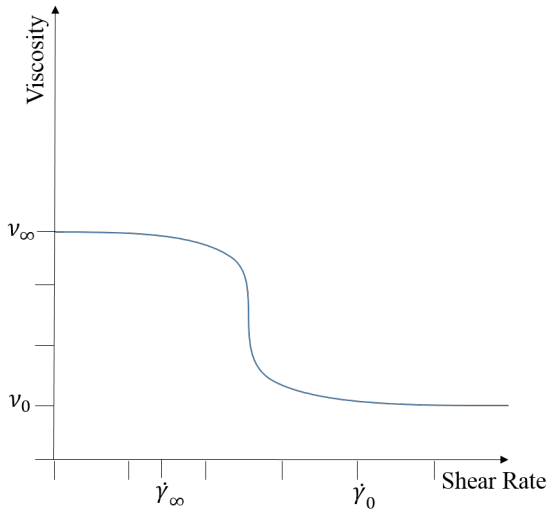

Figure 2.3: Viscosity versus Shear rate for a shear thinning fluid

The shear rate is defined as,

$$\dot{\gamma} \;=\; \sqrt{2\mathbf{d} : \mathbf{d}}, \tag{2.12}$$

where $\mathbf{d}$ is the strain rate tensor is,

$$\mathbf{d} \;=\; \frac{1}{2}\left(\nabla\mathbf{v} + \nabla\mathbf{v}^t\right). \tag{2.13}$$

2.4 Power Law Fluids

The Fig. 2.3 depicts viscosity decreasing with shear rate as shear thinning or pseudoplastic behavior. Most such fluids show similar relationships with Newtonian plateaux at very low and high shear rate. Here the viscosity is essentially constant given by ν_0 and ν_∞. ν_0 is the viscosity of the fluid at lower shear rates and ν_∞ is the viscosity at higher shear rates. In the region corresponding to intermediate shear stress, the $\log - \log$ graph is approximately linear. This implies that the behavior can be described as a power-law relationship. Such fluids are correspondingly called power law fluids.

Here we show some of the power law models used to fit viscometric data, as detailed in [4].

2.4.1 Cross Model

The cross model is a four constant model displaying non-zero bounded viscosity at lower and upper limits. It is represented by,

$$\frac{\nu - \nu_\infty}{\nu_0 - \nu_\infty} \;=\; \frac{1}{1 + (K\dot{\gamma})^m}. \tag{2.14}$$

Here, ν_0 and ν_∞ are, as earlier, asymptotic values of viscosity. K and m are constants with dimensions *time* and *naught* respectively. A fluid starts to show predominantly Newtonian behavior as $m \to 0$. the model fits experimental data for *aqueous polyvinyl acetate dispersion* and *aqueous limestone suspension*.

Another model which is similar to the Cross model at shear rates corresponding to asymptotic viscosity values is the Carreau model.

2.4.2 Carreau model

A generalization of Carreau model is the *Carreau-Yasuda model* where the denominator of m_1 is itself a variable parameter. This is represented as,

$$\frac{\nu - \nu_\infty}{\nu_0 - \nu_\infty} \;=\; \frac{1}{1 + ((K_1\dot{\gamma})^2)^{m_1/2}}. \tag{2.15}$$

Rearranging,

$$\nu \;=\; \nu_\infty + (\nu_0 - \nu_\infty)(1 + ((K_1\dot{\gamma})^2)^{\frac{m_1-1}{2}}). \tag{2.16}$$

The model fits the experimental data for *molten polystyrene*. It differs from the Cross model when $(K_1\dot{\gamma}) \to 1$.

$$\frac{\nu - \nu_\infty}{\nu_0 - \nu_\infty} \;=\; \frac{1}{1 + ((K_1\dot{\gamma})^a)^{m_1/a}}. \tag{2.17}$$

The relation for Ostwald de Waele model may be derived from the Cross model. Consider, if $\nu << \nu_0$, the Eq. 2.14 reduces to,

$$\nu \;=\; \nu_\infty \;+\; k\dot{\gamma}^{n-1}, \tag{2.18}$$

where $k = \nu_0 K^{-m}$ and $n = 1 - m$. Additionally, if $\nu >> \nu_\infty$, it further reduces to the Ostwald de Waele relation.

$$\nu \;=\; k\dot{\gamma}^{n-1} \tag{2.19}$$

2.4.3 Ellis Model

The models described above have all been of the form $\tau_{yx} = f(\dot{\gamma}_{yx})$. The Ellis Model is a departure from that rule, in that it gives the viscosity as a function of shear stress. This is given as following,

$$\nu = \frac{\nu_0}{1 + (\tau_{yx}/\tau_{1/2})^{\alpha-1}} \tag{2.20}$$

$\tau_{1/2}$ is the model parameter denoting the value of shear stress at which the apparent viscosity has reduced to $\nu_0/2$ and α is the measure of shear thinning behaviour. The form of equation allows one to calculate the velocity profile from the representative value of stress distribution.

Other models, such as *Sisko, Barus, Bingham Herschel-Bulkley*, etc., exist, but aren't as popular as the aforementioned and are also beyond the scope of this work.

2.5 Time Dependent Non-Newtonian Fluids

The models described till now did not take into account the duration for which shearing was applied to the fluid. There are certain fluids for which the rate of shear as well as the time duration contributes to the non-Newtonian behaviour. Examples of such fluids include crude oil, building materials, certain food stuffs etc. These exhibit non-Newtonian progressively as their internal structural linkages break down over time due to shear. As the number of internal structures breaking increases, after a point of time when there are no more links to be broken, the rate of change of viscosity also drops to zero. There might be a situation when the rate of

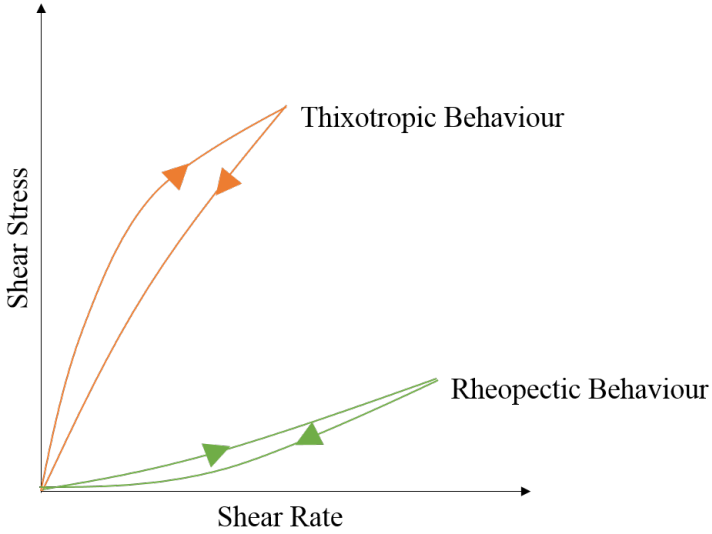

Figure 2.4: Time Dependent Non-Newtonian Fluid Behavior

breakage equals the rate of formation of the linkages leading to a dynamic equilibrium being reached between the two which will deter any viscosity change [3]. These fascinating substances are further divided into two kinds, namely, substances showing thixotropy and those showing rheopexy, or negative thixotropy (Fig. 2.4). However, we would not be looking at these in our work here.

2.6 Conclusion

We showed, in this chapter, the difference between Newtonian and non-Newtonian fluids. Although, the former are easy to understand and model, they form a microscopic minority of all the fluids present in and around us. Its their behavior with the application of varying shear stress which differentiates the two. Non-Newtonian fluids are further classified into dilatant, and pseudoplastic (which together with Newtonian are referred to as Gengeralised Newtonian Fluids), and, viscoplastic and Bingham plastic fluids. We also talk of various mathematical models for generalised Newtonian fluids and mention other fluid phenomenon as thixotropy and rheopexy.

Chapter 3

Physics behind fluids

Anything with a tendency to flow is fluid. In other words the inability of a substance to maintain shear stress for any length of time characterizes it as a fluid. In the succeeding sections we give an account of the traditional way to solve for properties of the fluid (i.e. Navier Stokes' equations). We also then introduce Lattice Boltzmann Method as an alternative to the conventional macroscopic view of fluid dynamics. Then, we show through Chapman Enskog Analysis how LBM converges to Navier Stokes' at the macro level.

3.1 Navier Stokes' Equations

We, as fluid animators, get introduced to the basics of fluid flow by the Navier Stokes' equations. In most cases, these equations are sufficient to model or accurately approximate real world behaviour which we want to simulate. Some times they don't, which we shall come to later. The Navier Stokes' equations, are,

$$\frac{\partial \mathbf{v}}{\partial t} \; + \; \mathbf{v}.\nabla\mathbf{v} \; + \; \frac{1}{\rho}\nabla p \; = \; \mathbf{f}_{ext} \; + \; \nu\nabla.\nabla\mathbf{v}, \tag{3.1}$$

$$\nabla.\mathbf{v} \; = \; 0, \tag{3.2}$$

where, $\mathbf{v}, t, \rho, p, \mathbf{f_{ext}}, \nu$ are the velocity field, time, density, pressure field, net external force (like gravity etc.) and viscosity, respectively. In the succeeding paragraphs we try to arrive at this set of equations from first principles. Consider an imaginary cube of liquid existing in the bulk of liquid. The cube is in equilibrium and is acted upon by an external force. For the cube with liquid having density ρ, pressure p and potential ϕ the equilibrium condition is given by,

$$-\nabla p \; - \; \rho\nabla\phi \; = \; 0 \tag{3.3}$$

This is simply balancing the forces acting on the imaginary cube. We conjecture that the density of the fluid ρ is constant throughout the fluid because if it were not so, then there

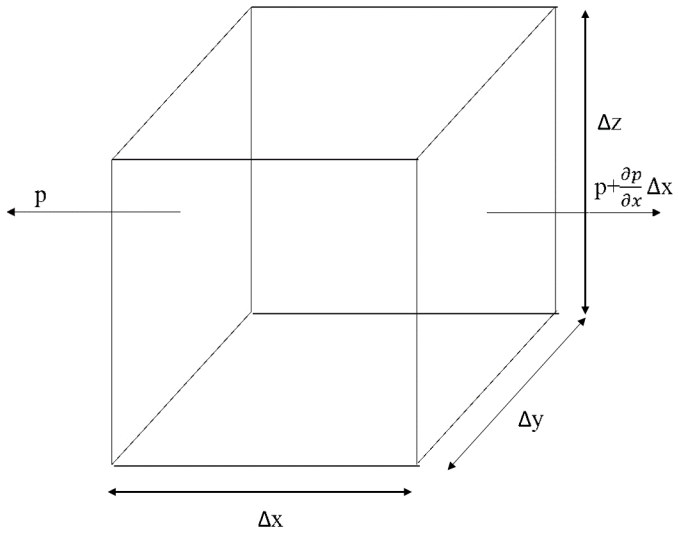

Figure 3.1: Force exerted over a fluid element due to the pressure exerted by the surrounding fluid

wouldn't be any way the forces would balance and it won't be in the state of equilibrium. Hence the above equation has only one solution namely,

$$p \ + \ \rho\phi \ = \ constant \tag{3.4}$$

We will continue to assume that our fluids are of constant density throughout this discussion. Let us now move to fluid in motion. Motion of fluids is physically very different from rigid bodies, in that, while the motion of the latter is fixed (a rigid body moving from one point to another would stop occupying space in the first point and move to the second), movement of the former is not. Hence the motion of a fluid is best described by describing its properties at every point in the simulation domain as a function of time. Thus, its motion can be best described by treating its properties as field. So, properties like velocity which will be different at each point at each time can be modeled as a vector field existing in the simulation domain. Density, pressure, etc., can similarly be modeled.

Let us consider conservation of mass in our flowing fluid. Consider a cube of unit volume through which the fluid flows. If the velocity of the fluid is $\mathbf{v}$ then the mass which flows through the surface of unit area in unit time is the component of $\rho\mathbf{v}$ which is normal to the surface. The total change in mass inside the volume element will be $-\frac{d\rho}{dt}$.

$$\nabla.(\rho\mathbf{v}) \ = \ -\frac{\partial\rho}{\partial t} \tag{3.5}$$

But, since density is constant,

$$\nabla.\mathbf{v} \ = \ 0. \tag{3.6}$$

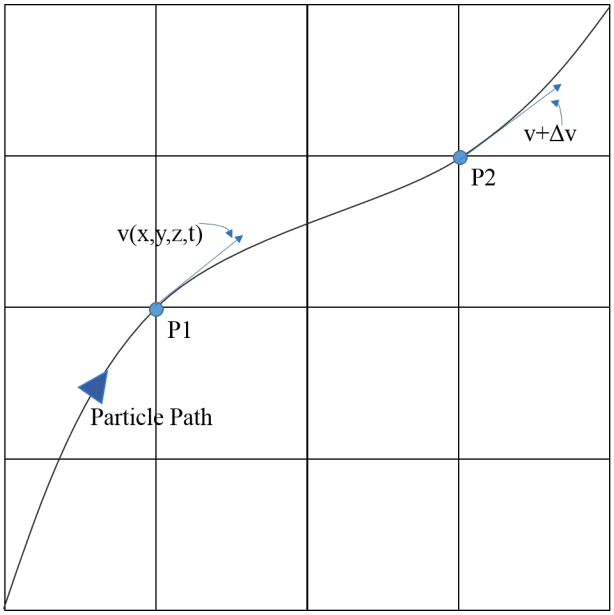

Figure 3.2: Path taken by a fluid element

This is also known as *continuity equation* in hydrodynamics and is the first Navier Stokes' equation. In a layman's terms it tells us that the amount of fluid entering a volume element is equal to the amount of fluid leaving it. The mass of fluid present in the element remains conserved. After mass conservation, we move towards momentum conservation. The total force applied to the cube of unit volume is equal to the product of the density with the acceleration produced.

$$\rho \; . \; accelaration \; = \; \mathbf{f}_{total} \tag{3.7}$$

Above we considered pressure force per unit volume when we talked of fluid in equilibrium. We computed the total force acting on the volume element in equilibrium as being $-\nabla p \; - \; \rho \nabla \phi$. Since the case of moving fluid is different by virtue of it being in motion, we add another term, which we call the viscous force. Hence,

$$\rho \; . \; acceleration \; = \; -\nabla p \; - \rho \nabla \phi \; + \; \mathbf{f}_{visc} \tag{3.8}$$

When we model velocity as a vector field denoted by $\mathbf{v}$ we cannot replace acceleration in the above equation by $\frac{\partial \mathbf{v}}{\partial T}$. $\frac{\partial \mathbf{v}}{\partial t}$ is another way of representing $\frac{\partial v}{\partial t}|_{x=constant}$ which means that it is the rate of change of velocity at a fixed point in space. When we talk of acceleration of a volume element we mean the acceleration of that element as it travels from one point to another. Let there be two adjoining points on a grid, $P1$ and $P2$. Let the velocity of the cube at $P1(x, y, z)$ be $v(x, y, z, t)$. Let the cube reach $P2(x + \triangle x, y + \triangle y, z + \triangle z)$ at $t + \triangle t$ be $\mathbf{v}(x + \triangle x, y + \triangle y, z + \triangle z, t + \triangle t)$ with

$$\triangle x \; = \; v_x . \triangle t, \; \triangle y \; = \; v_y . \triangle t, \; \triangle z \; = \; v_z . \triangle t \tag{3.9}$$

This implies that,

$$\mathbf{v}(x+v_x\triangle t, y+v_y\triangle t, z+v_z\triangle t, t+\triangle t) \;=\; \mathbf{v}(x,y,z,t) + \frac{\partial \mathbf{v}}{\partial x}v_x\triangle t + \frac{\partial \mathbf{v}}{\partial y}v_y\triangle t + \frac{\partial \mathbf{v}}{\partial z}v_z\triangle t + \frac{\partial \mathbf{v}}{\partial t}\triangle t. \tag{3.10}$$

Rearranging the terms we find that,

$$\frac{\mathbf{v}(x+v_x\triangle t, y+v_y\triangle t, z+v_z\triangle t, t+\triangle t) \;-\; \mathbf{v}(x,y,z,t)}{\triangle t} \;=\; (\mathbf{v}.\nabla)\mathbf{v} + \frac{\partial \mathbf{v}}{\partial t} \tag{3.11}$$

The LHS signifies the acceleration of the volume element. Replacing this result in our Newton's second law equation we find that,

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}.\nabla).\mathbf{v} = -\frac{\nabla p}{\rho} \;-\; \nabla\phi \;+\; \frac{\mathbf{f}_{visc}}{\rho} \tag{3.12}$$

which is the second Navier Stokes' equation. The operator,

$$(\mathbf{v}.\nabla) \;+\; \frac{\partial}{\partial t} \tag{3.13}$$

is called the *material derivative* and is represented as $\frac{D}{Dt}$.

3.2 Mathematical Foundation of Lattice Boltzmann Method

LBM, introduced as a statistical approach to model fluids, takes an approach different from both molecular dynamics and methods based on discretization of partial differential equations including, but not restricted to, finite difference, finite element etc. Although Lattice Boltzmann Method (LBM) is the discretized view of the continuous Boltzmann equation in Kinetic Theory, it was conceived as a logical progeny of Lattice Gas Cellular Automaton.

3.2.1 Lattice Gas Cellular Automaton

That different mesoscopic interactions can, when aggregated, conform to the given set of macroscopic equations is the starting point of Lattice Gas Cellular Automaton. Consider a fixed grid spanning the simulation domain, with logical particles constrained to move along the grid connections. Particles are assumed to be unit mass. Each center in the nexus is called a cell. A cell can be occupied by at most one particle. Since only one bit is sufficient to describe a cell-state (occupied or empty), hence the name, cellular automaton. Assuming unit time steps, the grid connections give the lattice velocities. At each time step, particles interact at a grid center, undergo momentum changes while conserving the total momentum. They are then, propagated or streamed further along the associated link to the next grid center. The interaction of particles and their redistribution at each grid centre is termed collision.

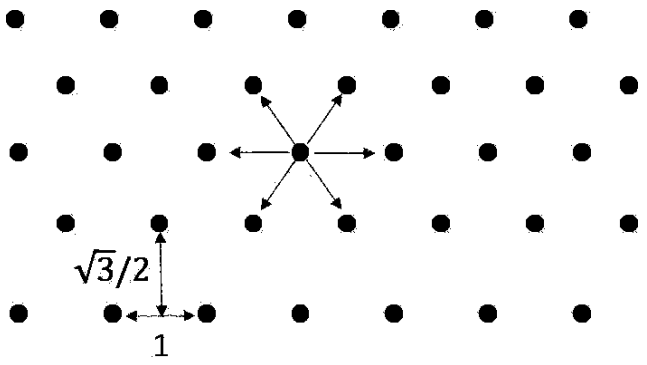

Figure 3.3: FHP Lattice (hexagonal symmetry). Arrows are the lattice velocity vectors c_i.

It has been established that only some lattice configurations can work to model fluid behaviour. These have to have certain symmetry associated with them. Thus, the square symmetrc model proposed by Hardy, Pomeau and de Pazzis (HPP) is not able to model fluids accurately. However a regular hexagonal (or triangular, depending on how one looks at it) lattice, comprising the Frisch, Hasslacher, and Pomeau (FHP) model with each node having six neighbours, can. Thus, at each node, six cells are present which may be occupied by at most a particle of unit mass (for ease in computation). The state of a node, therefore is defined by six bits. The node connection vectors, connecting each node to its neighbours, c_i, are as follows,

$$c_i \;=\; \left(\cos\frac{\pi}{3}i\;,\;\sin\frac{\pi}{3}i\right), \quad i \;=\; 0,\; ...,5. \tag{3.14}$$

These are associated with each cell and are also termed lattice velocities, since the time step is assumed to be unity. Since we have unit mass particles, these are also the particles' momenta. The evolution of the simulation happens with successive collision and propagation (streaming) steps during each time step. While propagation happens, for each cell, along its lattice vector to the corresponding cell in the neighbouring node, collision is a strictly local process. Particles in the cells for a node participate in collision according to the rules enunciated in fig. (3.4).

Each lattice centre is described by six (in case of FHP) bits or six occupation numbers $n_i(t, \mathbf{r}_j)$ where i gives the direction of each occupation number. The numbers themselves may be zero or one. The mean occupation number $N_i(t, \mathbf{x})$ is therefore,

$$N_i(t, \mathbf{x}) = \langle n_i(t, \mathbf{r}_j)\rangle. \tag{3.15}$$

Since each particle is of unit mass, the mass density and the momentum density $\rho(t, \mathbf{x})$ can be calculated as,

$$\rho(t, \mathbf{x}) \;=\; \sum_{i=0}^{5} N_i(t, \mathbf{x}) \tag{3.16}$$

$$j(t, \mathbf{x}) \;=\; \sum_{i=0}^{5} c_i N_i(t, \mathbf{x}). \tag{3.17}$$

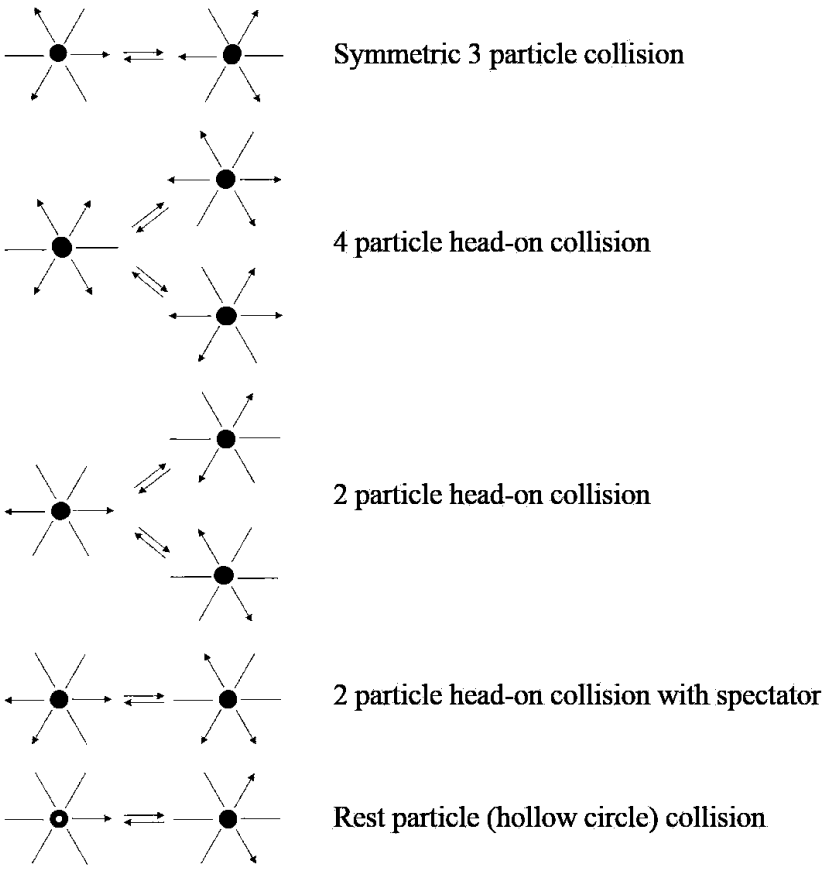

Figure 3.4: FHP Lattice (hexagonal symmetry). Arrows are the lattice velocity vectors c_i.

These values can be coarse grained further over reasonable sub-domains to obtain reliable (low noise) averages. We direct the reader to [5] for a more comprehensive explanation of the same.

3.2.2 Dynamics of the Lattice Boltzmann Method

The Kinetic theory models fluids on the molecular level by describing the non-equilibrium process within the gasses and the path the gas takes to achieve thermodynamic equilibrium. *Equilibrium*, here, would be the state where, at least *at the coarse grained level, things don't change.*

The model takes into consideration the velocity and position of each particle in the gas. Each particle is treated as point-like. It states that a fully exhaustive information of the system can be obtained out of $6N$ functions $[\mathbf{x}_i(t), \mathbf{p}_i(t)]$, with $i = 0, 1, 2, ..., N-1$ in Euclidean space. The development of the model takes into consideration only bi-particular collisions and assumes the collisions to be unaffected by any external force. The collisions are thus localised with the particles spending most of their free time on free trajectories. Based on these principles, the

Maxwell-Boltzmann probability distribution is defined as

$$f(\mathbf{v})d^3v = \left(\frac{m}{2\pi k_B T}\right)^{3/2} e^{-mv^2/2k_BT} d^3v \tag{3.18}$$

Where LHS is the probability that a particle has velocity within a small volume d^3v in the neighbourhood of $\mathbf{v}$. The quantity f is called the single particle velocity distribution function. The statistics can also be defined in terms of the single particle phase space distribution function which essentially gives the probability of the particle being at a location $\mathbf{x}$ having velocity $\mathbf{v}$, or,

$$f(\mathbf{x}, \mathbf{v}, t) \;=\; \frac{dN(\mathbf{x}, \mathbf{v}, t)}{d\mathbf{x}d\mathbf{v}}. \tag{3.19}$$

Thus $\frac{dN}{d\mathbf{x}d\mathbf{v}}$ represents the probable number of particles at $\mathbf{x}$ with velocity $\mathbf{v}$ at time t.

The Kinetc equation for one body distribution function is,

$$\left[\partial_t \;+\; \frac{\mathbf{p}}{m}.\partial_{\mathbf{x}} \;+\; \mathbf{F}.\partial_{\mathbf{p}}\right] f(\mathbf{x}, \mathbf{p}, t) \;=\; \Omega_{12} \tag{3.20}$$

The left hand side represents the streaming motion of the particles in the presence of an external force field $\mathbf{F}$. On the right hand side is the effect of *interparticular collisions*. The suffix 12 is the result of assuming localised binary collisions. The quantity Ω_{12} is called the *collision operator* and involves the probability of finding particle 1 at $\mathbf{x_1}$ with velocity $\mathbf{v_1}$ and particle 2 at $\mathbf{x_2}$ with velocity $\mathbf{v_2}$, at time t. More precisely, if the two colliding particles are characterized by distribution functions f_1 and f_2 before collision and f_1' and f_2' after collision then the collision operator is,

$$\Omega_{12} \;=\; \int (f_{12}' \;-\; f_{12}) g\sigma(g, \phi) d\phi d\mathbf{p_2} \tag{3.21}$$

Where σ is the differential cross section expressing the number of particles with relative velocity $\mathbf{g} \;=\; \mathbf{v_1} - \mathbf{v_2}$ and ϕ is the solid angle defined for the collision.

Since we do not assume any correlation between the two colliding particles, this gives rise to the *closure assumption*, viz.,

$$f_{12} \;=\; f_1.f_2 \tag{3.22}$$

With this, we now define the local equilibrium distribution function, which when taken as the distribution function for the two colliding particles vanishes the collision term.

$$\Omega(f^e, f^e) = 0. \tag{3.23}$$

This is due to the *detailed balance condition* which equates the post-collision distribution functions to the pre-collision distribution functions, all of which are equal to the equilibrium value of the distribution function, i.e.,

$$f_1' f_2' \;=\; f_1 f_2 \tag{3.24}$$

Intuitively, this can be understood as the equilibrium state where the properties and hence the behaviour of the particles do not change. It should be noted here that the equilibrium state doesn't mean that the particles sit idle. It just means that at equilibrium, the effect of collision balances out.

The nature of the collision operator is complex and therefore contradicts the otherwise easy understanding and implementation of Lattice Boltzmann method. It is hence more often than not replaced by simpler approximations which nevertheless preserve collision invariants. The BGK method named after and proposed by Bhatnagar *et al.* [6] is used which is,

$$\Omega_{BGK}(f) \;=\; \frac{f^e - f}{\tau}, \tag{3.25}$$

where, τ is the *relaxation time*, i.e., the time between two successive collisions experienced by the particles. The density ρ, momentum $\mathbf{p}$ and energy E, assuming a particle mass of unity, can be computed as,

$$\int f d\mathbf{v} \;=\; \rho \tag{3.26}$$

$$\int f v_a d\mathbf{v} \;=\; \rho u_a \tag{3.27}$$

$$\int f \frac{v^2}{2} d\mathbf{v} \;=\; \rho E \tag{3.28}$$

u_a is the macroscopic flow speed whereas v_a is the velocity of the particle. We are now in a position to derive the *Lattice Boltzmann Equation* from the *continuous Boltzmann Equation*. The continuous Boltzmann Equation in the absence of an external force field and after approximating the collision operator will take the form,

$$\left[\partial_t \;+\; \mathbf{v}.\partial_{\mathbf{x}}\right] f(t) \;=\; -\frac{1}{\lambda}\left(f(t) \;-\; g(t)\right)) \tag{3.29}$$

We represent the above equation as an ordinary differential equation,

$$\frac{Df}{Dt} \;+\; \frac{f}{\lambda} \;=\; \frac{g}{\lambda} \tag{3.30}$$

where,

$$\frac{D}{Dt} \;=\; \frac{\partial}{\partial t} \;+\; \mathbf{v}.\nabla \tag{3.31}$$

Eq. 3.30 is a linear first order ordinary differential equation. The solution for it is therefore,

$$f(t+\delta_t) \;=\; f(t).e^{-\frac{\delta_t}{\lambda}} + \frac{1}{\lambda} e^{-\frac{\delta_t}{\lambda}}.\int_0^{\delta_t} e^{\frac{t'}{\lambda}} g(t+t')dt' \tag{3.32}$$

We can obtain an interpolated value of $g(t+t')$ for $0 \leq t' \leq \delta_t$ if it is assumed that $delta_t$ is very small and g is a smooth funtion.

$$g(t+t') \;=\; \left(1 - \frac{t'}{\delta_t}\right) g(t) \;+\; \frac{t'}{\delta_t} g(t+\delta_t) \;+\; O(\delta_t^2) \tag{3.33}$$

Applying Eq. 3.33 in Eq. 3.32,

$$\frac{1}{\lambda}e^{-\frac{\delta_t}{\lambda}}.\int_0^{\delta_t} e^{\frac{t'}{\lambda}}g(t+t')dt' = \frac{1}{\lambda}e^{-\frac{\delta_t}{\lambda}}.\int_0^{\delta_t} e^{\frac{t'}{\lambda}}\left(\left(1-\frac{t'}{\delta_t}\right)g(t) \;+\; \frac{t'}{\delta_t}g(t+\delta_t)\right)dt'$$

$$= \frac{1}{\lambda}e^{-\frac{\delta_t}{\lambda}}\left[e^{-\frac{\delta_t}{\lambda}}\lambda g(t) \;-\; e^{-\frac{\delta_t}{\lambda}}\frac{\frac{1}{\lambda}t'-1}{\frac{1}{\lambda^2}}\frac{g(t)}{\delta_t} \;+\; e^{-\frac{\delta_t}{\lambda}}\frac{\frac{1}{\lambda}t'-1}{\frac{1}{\lambda^2}}\frac{g(t+\delta_t)}{\delta_t}\right]_0^{\delta_t}$$

$$= g(t) \;-\; e^{-\frac{\delta_t}{\lambda}}g(t) \;+\; \left[1 \;+\; \frac{\lambda}{\delta_t}\left(e^{-\frac{\delta_t}{\lambda}}-1\right)\right][g(t+\delta_t)-g(t)] \quad (3.34)$$

Putting Eq. 3.34 in Eq. 3.32,

$$f(t+\delta_t) \;- f(t) = \left(e^{-\frac{\delta_t}{\lambda}}-1\right)[f(t)-g(t)] \;+\; \left(1+\frac{\lambda}{\delta_t}\left(e^{-\frac{\delta_t}{\lambda}}-1\right)\right)[g(t+\delta_t)-g(t)] \quad (3.35)$$

Expanding $e^{-\frac{\delta_t}{\lambda}}$ as a Taylor Series

$$e^{-\frac{\delta_t}{\lambda}} \;=\; e^{-\frac{0+\delta_t}{\lambda}} \;=\; 1+\delta_t\left(-\frac{1}{\lambda^2}e^0\right) \;+\; O(\delta^2) \;\approx\; 1 \;-\; \frac{\delta_t}{\lambda} \quad (3.36)$$

Putting Eq. 3.36 into Eq. 3.35,

$$f(t+\delta_t) \;- f(t) = \left(1-\frac{\delta_t}{\lambda}-1\right)[f(t)-g(t)] \;+\; \left(1+\frac{\lambda}{\delta_t}\left(1-\frac{\delta_t}{\lambda}-1\right)\right)[g(t+\delta_t)-g(t)]$$

$$= \;-\frac{\delta_t}{\lambda}\left(f(t)-g(t)\right) \quad (3.37)$$

Replacing $\frac{\delta_t}{\lambda}$ with $\frac{1}{\tau}$ and $g(t)$ with $f^e(t)$ in Eq. 3.37, we arrive at,

$$f(t+\delta_t) \;-\; f(t) \;=\; -\frac{1}{\tau}\left[f(t)-f^e(t)\right] \quad (3.38)$$

3.2.3 Chapman Enskog Analysis

The goal is to start from Boltzmann equation and reach a set of partial differential equations in terms of ρ and $\rho\mathbf{v}$ which approximate the behaviour of a fluid as $dx \to 0$ and $dt \to 0$ with $\frac{dx}{dt} = constant$. The distribution function may be expanded as a Taylor series,

$$f \;=\; f^0 \;+\; \epsilon f^1, \quad where\ f^0 = f^e \quad (3.39)$$

The superscript zero depicts the equilibrium distribution function, and subsequent numerical subscripts, the departure from the equilibrium distribution function. Epsilon is generally taken to be the Knudsen number, which is the relation of the mean free path of the fluid to the characteristic length of the channel in which the fluid flows. For example, if the fluid is flowing in a cylindrical pipe then the diameter of the pipe is taken to be the characteristic length.

The Taylor series expansion points to a multiscale analysis. The idea is to represent the space and time variables in terms of a hierarchy of scales going from slow to fast or long to short, such that each variable is $O(1)$ at its own scale. The mutiscale variables are of the form,

$$x \;=\; \epsilon^{-1} x_1 \tag{3.40}$$

$$t \;=\; \epsilon^{-1} t_1 + \epsilon^{-2} t_2 \tag{3.41}$$

The time scale t is of the order of the time taken during collisions which is extremely small. Likewise, t_1 is the timescale of propagation of sound waves which is, in turn, faster than the time taken by phenomenon such as diffusion, whose timescale is represented as t_2. We consider only one scale for space. This when translated to differential operator terms is,

$$\partial_x \;= \epsilon \partial_{x_1} \tag{3.42}$$

$$\partial_t \;= \epsilon \partial_{t_1} \;+\; \epsilon^2 \partial_{t_2} \tag{3.43}$$

So, in order to achieve order by order consistency, we have to expand the Boltzmann equation up to the secoind order.

$$\frac{D}{Dt} \;\equiv\; D_t \;\sim\; \epsilon \partial_{t_1} \;+\; \epsilon^2 \partial_{t_2} + \epsilon v_a \partial_{x_{1_a}} \;+\; \frac{1}{2} \epsilon^2 v_a v_b \partial_{x_{1_a}} \partial_{x_{1_b}} \tag{3.44}$$

The collision operator, is also expanded as,

$$\Omega\left[f\right] \;\sim\; \Omega\left[f^0\right] \;+\; \epsilon \Omega' f^1 \tag{3.45}$$

Ω' denotes the functional derivative of Ω with respet to f. We know that the equilibrium distribution function vanishes the collision operator hence there is no effect of the first term on the RHS. The Boltzmann equation can be seen to be consisting of $O(\epsilon^0)$ $O(\epsilon)$ and $O(\epsilon^2)$ terms and are separately handled. These equations show that each term in the operational coefficients of the expansions should be zero. The mass and momentum conservation give rise to Eq. 3.46 and Eq. 3.47 respectively.

$$\epsilon \hat{M} \;+\; \epsilon^2 \hat{M} \;=\; 0 \tag{3.46}$$

$$\epsilon \hat{J} \;+\; \epsilon^2 \hat{J} \;=\; 0 \tag{3.47}$$

Where $J_a \;=\; \rho u_a$. This is explained in detail in Frisch *et al.* [7]. The first order terms $O(\epsilon)$, in the Boltzmann equation, using a mass and momentum of zero give rise to the following two equations,

$$\partial_{t_1} \rho \;+\; \partial_{a_1} J_a \;=\; 0, \tag{3.48}$$

$$\partial_{t_1} J_a \;+\; \partial_{b_1} \int v_a v_b f^0 d\mathbf{v} \;=\; 0. \tag{3.49}$$

While the first equation is evidently the continuity equation, the second equation is the Euler equation for inviscid flows without dissipation, if the integral is replaced by,

$$\rho u_a u_b \ + \ \rho T \delta_{ab}. \tag{3.50}$$

The Navier Stokes' equation can be obtained from here after considering the second order terms as well. Here, the complications increase due to the requirement of equilibrium and non-equilibrium terms. By setting the first order terms to zero and restoring the continuous form of the equations, we can obtain the Navier Stokes' equation (momentum equation). Fagerberg *et al.* [8] discuss the physics behind LBM in extensive detail.

3.3 Conclusion

In this chapter we saw in brief the laws of physics which work behind fluid motion. We saw a statistical method to model fluid i.e. Lattice Boltzmann Method and its predecessor, Lattice Gas Cellular Automaton. Although LGCA is governed by extremely simple rules of collision and propagation (coupled with symmetric lattice structure) making it very easy to implement on a computer, it suffers from rapidly increasing noise as the level of detail increases. LBM was an advancement over LGCA but used descretised Boltzmann Equation over the simple rules of the latter. It also maps propagation in terms of probability density rather than boolean particles. Finally we saw how Chapman Enskog Analysis proves that LBM equations at macro are nothing but Navier Stokes' Equations, thus confirming the accuracy of the method.

Chapter 4

Classic Methods for Fluid Simulation

Now that we are sufficiently acquainted with the kinds of fluids as well as what lies beneath the surface of Navier Stokes' equations, we, here, describe the history of fluid simulation in terms of the development of Eulerian and Lagrangian methods. These are the two most popular and well documented methods. BAsic Lagrangian/Eulerian simulations are the stepping stones for any programmer venturing into the domain for the first time.

The history of fluid simulations, thus, rightly, begins with the formulation of the Navier Stokes' equations developed in the 1840s. We now know they were a set of partial differential equations formed on the basis of conservation laws and first order approximations. What followed was the conventional study of fluid flows for more than a century. Arriving at computational models to solve fluid equations has been a subject of research since the early 1950s. Finding solutions to the partial differentials of Navier Stokes' equations using discrete algorithms was area of focus. Many modern day techniques, of which some will be skimmed through in the succeeding chapters, came up during that time. Staggered marker-and-cell (MAC grid structure), Particle in Cell (PIC method) etc. are two of those.

However, most of the models and techniques developed by the CFD community then was complex and unscalable for *visual effects oriented computer graphics*. In the succeeding years fluid effects was generated using non-physics based methods, such as using hand drawn animation (key frame animation) or displacement mapping. The seminal work of Foster *et al.* [1] proved to be the adrenalin shot of Computer Graphics. This was the begining of Eulerian fluid simulation method. Stam [2] was also one of the early contributors to the field. Since then, the fluid simulation engines have come a long way with engines now reproducing complicated behaviour with increasing ease. We shall look at a few techniques developed along the years in the succeeding sections.

4.1 Eulerian Simulation

Eulerian simulations model fluid in a fixed grid by calculating the change in field values at each time step at each grid point. Stam [2] gave an intuitive method to do so by marching through time t with a time step of δt. Each frame is broken down into four sequential time steps and the field value of the previous time step is fed into the pipeline. The final result is the value of the field at the start of the next time step. This is depicted by the equation,

$$w_0 \overset{\text{add force}}{\rightarrow} w_1 \overset{\text{advect}}{\rightarrow} w_2 \overset{\text{diffuse}}{\rightarrow} w_3 \overset{\text{project}}{\rightarrow} w_4 \tag{4.1}$$

where w_0 is the field value at point $\vec{x}$ at time t and w_4 at time $t+\delta t$. The process is called splitting, which means the splitting of a complicated equation to easier to solve mutually exclusive but sequential parts and solving each of them one after the other. The parts here, therefore, are *addition of external force (eg. gravity), advection, diffusion and projection.* Advection simply is the transport of fluid dromone part to another while projection means to apply pressure so as to make the field divergence-free and satisfy boundary conditions.

The Eulerian viewpoint is such that at each fixed point, the field values are computed as the fluid flows past it. It isn't very intuitive at the first glance. However, the method eases computations as they are done at fixed points rather than on points which are floating seemingly arbitrarily. Also, the use of spatial derivatives like pressure and velocity gradients is analytically easier.

The method can be intuitively understood by the example of measuring temperature in a room. The thermometer is present at a particular location in the room and therefore measures the temperature at that location. The changes occurring in the air as it rushes past the location would be measured at the point where the thermometer head is at. Thus we also measure the changes in the fluid fields at fixed points as the fluid rushes past those points. Subsequently we model the fluid based on these variables.

Since the method progresses by computing values at specific grid points, the structure of the grid naturally becomes an important consideration. The *marker-and-cell* (MAC) grid was one of the earliest solutions suggested for the problem. The MAC grid is *staggered.* This means rather than storing all variables at one point uniformly across the grid, different variables are stored at different locations. The scalar field values are generally stored at each cell center while the vector field values are split into their components and are stored in the cell faces. Fig. (4.1) and Fig. (4.2) show the storing of variable values at faces and cell centers in two and three dimensions respectively.

As is evident from the figures, the velocity field values cannot be used as they are stored in the grid structure. So as to arrive at the field value at a grid cell, central difference of the values stored at the faces needs to be computed. This seems to be an overkill until one realizes that the partial derivatives required in Navier Stokes' computations need central difference as

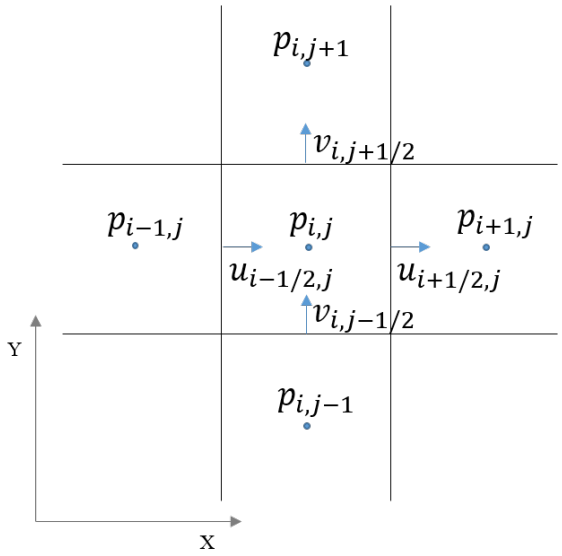

Figure 4.1: 2D MAC Grid

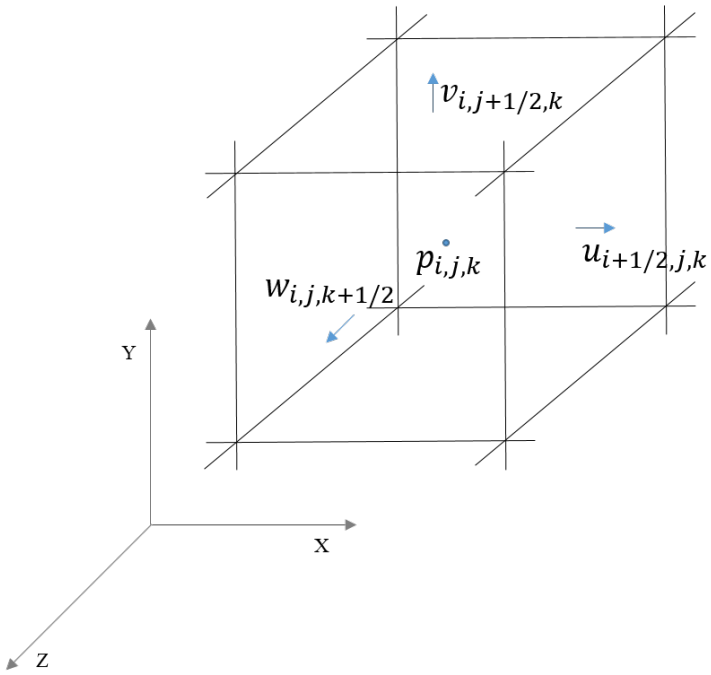

Figure 4.2: 3D MAC Grid

well, thus justifying the storage of velocity values at the faces and not at the cell centers. We shall revisit the method in Chapter 4.

4.2 Lagrangian simulation

The Lagrangian view is generally characterized by two kinds of methods, Smoothed Particle Hydrodynamics (SPH) and Discrete Vortex Method (DVM). These are also called *mesh free discretizations*. The particles are tracked as they move through the fluid field. It's important to note that these particles are not particles of the fluid, rather, they are an irregular discretization of the continuum - lumps carrying fluid information with them. At any given point if one has to find the fluid property one uses interpolation between the particles. Theoretically, each particle exerts an influence on the other particles. However in SPH based models, a smoothing function (kernel) is used to interpolate and threshold particular influence in it's neighborhood. These methods work very well splashes, sprays, jets etc. They are also more intuitive than Eulerian as they conform to the idea of tracking particles rather than abstract fields.

Typically, it begins with initializing N particles with masses m_i, positions x_i, velocity v_i and external force f_i $(0 \leq i \leq N-1)$. Since these are liquid particles, each particle exerts a force on every other particle, accounting for $O(N^2)$ interactions. This is unmanageable even when dealing with a relatively few particles, say 10000. This comes to a hundred million interactions in real time. So to make the simulation practical, we assume that the sphere of influence of each particle is not infinite but is restricted to a particular distance. We also need to ensure that the force exerted, though dwindling with distance, is C^0 and C^1 continuous at the cutoff distance, say d.

Thus, we track discrete particles and interpolate smooth fields out of them. To calculate continuous fields from discrete samples, we use a smoothing kernel $W(r)$. It defines a weighting function in the neighborhood of particle at x_i. As it is direction invariant, it has to be symmetric. It obviously, also needs to be normalized ($\int_0^d W(\mathbf{x}-\mathbf{x_i})dx = 1$). A popular choice for the same is the *poly6* kernel.

$$W(r) = \frac{315}{64\pi d^9} \begin{cases} (d^2 - r^2)^3 & 0 \leq \mathrm{r} \leq \mathrm{d} \\ 0 & \text{otherwise} \end{cases}, \tag{4.2}$$

The density function can thus be found by using the following function.

$$\rho(\mathbf{x}) = \sum_j m_j W(|\mathbf{x} - \mathbf{x_j}|) \tag{4.3}$$

The smoothed field values can be computed using the density and the smoothing kernel.

$$A_S(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{x} - \mathbf{x_j}|) \tag{4.4}$$

Through this, the gradient of the field is also easily calculated.

$$\nabla A_s(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|\mathbf{x} - \mathbf{x_j}|) \tag{4.5}$$

Since, we do care about tracking particles in the method, the LHS of the Navier Stokes' equation can be replaced by the material derivative. Since the particles of the fluid would move along with the fluid the material derivative would be equivalent to the time derivative of the velocity of the particles. Similarly for the RHS, ∇p would be clculated as follows.

$$\nabla p(\mathbf{x_i}) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(|\mathbf{x_i} - \mathbf{x_j}|) \tag{4.6}$$

The viscous force, given by $\mu \nabla^2 \mathbf{u}(\mathbf{x_i})$ is given by,

$$\mu \nabla^2 \mathbf{u}(\mathbf{x_i}) = \mu \sum_j m_j \frac{\mathbf{v_j} - \mathbf{v_i}}{\rho_j} \nabla^2 W(|\mathbf{x_i} - \mathbf{x_j}|). \tag{4.7}$$

In Eq. 4.7, the difference of velocities is used to bring symmetry in the viscous force, since the force would depend anyway on the relative and not absolute viscosities. The external forces like gravity can be tackled on a per particle basis since they don't affect the Navier Stokes' behavior internally.

Due to its mesh-less nature, Lagrangian methods can tackle multiple fluid interaction comparatively easily and are generally less resource intensive. However it has been difficult for Lagrangian methods to match the level of detail in free surface reconstruction generated by Eulerian methods.

4.3 Previous Works

As mentioned earlier, before the latter half of 1990s, fluid animation was either hand drawn or used bump mapping tricks or yet other non-physics based hacks. CFD models did exist, but were (often) needlessly complex and had poor scalability. Foster and Metaxas did pioneering work in "free surface flow" in [1] in 1996 using the standard MAC grid. Joe Stam in [2], took their method (Eulerian) forward by devising a technique which was *semi-Lagrangian* in nature. In this, advection entailed obtaining the velocity at a point x at the new time $t + \delta t$ by backtracing the point x through the velocity field over a time δt. These earlier methods suffered from non-conservation of sub-grid mass. Enright *et al.* [9] solved the problem by inventing *Particle Level Sets*. They later followed it up in Enright *et al.* [10]. The method was utilised for not just liquid simulations but also for simulation of turbulent gasses, for example, in Nguyen *et al.* [11]. Particle Level Sets can be used not just for Eulerian simulations but for Lagrangian as well, as was shown in Losasso *et al.* [12]. Eulerian simulations often have difficulty in producing small scale effects like sprays, foam, etc., which are essentially sub-grid

in nature. In such cases, Lagrangian simulations prove beneficial. In spite of this, Eulerian methods have remained relevant even in recent times as can be shown by Lentine *et al.* [13], which tries to rectify the problem of mass and momentum conservation of semi-Lagrangian advection. (Ronald Fedkiw has been an eminent contributor to the field and was awarded the Academy Award for Technical Achievement from The Academy of Motion Picture, Arts and Sciences.)

Conservation of mass and momentum was found lacking in earlier Eulerian methods. Since such methods were dependent on a skeletal grid structure, they were constrained by the grid itself to a minimum resolution, in terms of effects simulated. The minute effects which were sub-grid in size, such as foam, could therefore not be reproduced. Some of these were rectified in later times to an extent by using adaptive grids. However, Lagrangian methods were developed earlier to counter the shortcomings of Eulerian simulation. Since they deal with particles, they make mass conservation and convection terms used in the standard Eulerian models redundant and therefore dispensable. The particles simulated can either be directly used to render the surface of the fluid or, meshing techniques such as Marching Cubes can be used to render. Smooth Particle Hydrodynamics is a method which was used to solve astrophysical problems before finding usage in fluid modelling. Desburn at al.'s efforts in this regard deserve mention. They in Desbrun *et al.* [14] presented the method as a means to simulate highly deformable bodies as particle systems. Muller *et al.* in [15] carried this method forward to simulate free surfaces.

Hybrid methods such as (Fluid Implicit Particle) FLIP or Material Point Method (MPM) have become popular nowadays in the industry. The FLIP method was formulated in 1986 and was dependent on the particles which carried all the information necessary to describe the fluid. Using the particle data, the Lagrangian moment equations are solved on an adaptive (preferably) grid. The FLIP was studied and carried forward by Bridson *et al.* in [16].

Over the years the complexity of the simulations has also increased. Meshing algorithms have also been developed in keeping with the increase in demand for detail in free surface flows. Marching Cubes have already been mentioned as an easy and comparatively inexpensive way for mesh generation. Particle Level Sets improved the quality further. Recent developments include Lattice based Tetrahedral Meshes, as described in Chentanez *et al.* [17].

The interest in application of statistical models to fluid simulation started in the 1970s. The methods enumerated above look at fluid mechanics from the top. This means that they deal with the Navier Stokes' equation, which describes the macroscopic behaviour of the fluid and not how that behaviour is a result of particular interaction at a finer level. The statistical approaches did the opposite. Lattice Gas Cellular Automata was the pioneering work in this direction. The LGCA is a discrete model comprising a grid system with evolution based on mathematical rules. This was based on the fact that *for a grid exhibiting a specific symmetry*,

if the particular interactions happening on the grid follow the basic rules of mechanics viz. conservation of mas, momentum and energy, the expected fluid behaviour can be observed.

Hardy, de Pazzis and Pomeau in 1973 came up with the HPP LGCA. This described a square symmetry grid with rules of evolution. The model conserved mass and momentum, however, it wasn't symmetric enough as is explained in later sections. 1986 saw Frisch, Haslachar and Pomeau come up with the FHP lattice. The lattice structure was hexagonal as it was found that in two dimensions a degree-four symmetry wasn't enough whereas hexagonal symmetry was. Symmetry ensured the isotropy of a certain rank-four tensor formed by the lattice velocities. This effectively solved the problem with LGCA methods in two dimensions, however they still suffered from the statistical noise issues post coarse graining. Although LBM is the discretization of the Boltzmann equation, it was developed from LGCA. Where LGCA deals with unit particles, LBM deals with distribution functions. It doesn't need coarse graining. The interactions (termed "collisions") involved are local. Propagations happen accordingly after collisions. The method was first described by Chen at al. in [18]. Nils Thuerey has been on the forefront of developing LBM including Thuerey *et al.* in [19] and [20] where he developed a method for simulating free surface flows using the LBM. He optimised the grid structure to be adaptive. This was followed by Thuerey *et al.* [21] where interaction of a LBM fluid was discussed with deformable boundaries. The method was further improved upon by inculcating adaptive time steps and an adaptively coarsened grid structure Thuerey *et al.* [22]. The method performs better than both LGCA and conventional methods in that it deals with the problem of statistical noise satisfactorily and is also accurate to the second order, in contrast with the conventional methods which display first order accuracy.

Over the last decade and a half various niche problems concerning non-Newtonian fluids have been tackled. Efforts have gone in developing methods in the applied mathematics field and to an extent graphics. Goktekin *et al.* [23] dealt with viscoelastic fluids, i.e., the fluids exhibiting both viscous (characteristic of liquids) and elastic (characteristic of solids) properties. The implementation is essentially Eulerian on a staggered MAC grid proposed by Harlow *et al.* [24] a Lagrangian approach towards viscoelastic simulation was taken. Boyd *et al.* [25], Gabbanelli *et al.* [26] have presented models for power law fluids using the LGBK model. Wagner *et al.* have presented a MRT (Multi Relaxation Time) model for fluid flows in Kaehler *et al.* [27].

Chapter 5

Lattice Boltzmann Method on a computer

We covered the physics behind Lattce Boltzmann Method in Ch. 3. Through Chapman Enskog Analysis, we proved that it does lead to Navier Stokes' behaviour at the macro level. In this chapter we further we show how Lattice Boltzmann Method can be implemented on a computer for GNF.

The Lattice Boltzmann Model is the discretised representation of the Boltzmann Model of the Kinetic Theory of Gasses. Hence we begin by discretizing the space in a *cartesian* fashion. The particles, logical in nature, are constrained to move in a fixed number of directions from one grid cell to another. Apart from this they show regular Boltzmann behavior. Since particular interactions are more intuitive than abstract wave based physics followed by macroscopic top down approaches, LBM proves easier to understand. The particular interactions themselves are not complex and happen in regular discrete steps on an equidistant Cartesian grid. This makes for their easy implementation on the computer as well. The necessary fluid properties such as velocity, density etc., are obtained from coalescing properties carried by individual particles in each grid.

The grid is divided into individual cells with each cell having a certain number of particles constrained to travel in fixed directions. These may be 9 (for two dimensional simulations), 15, 19, 27 (for three dimensions) etc. The choice for the number of directions is made on the basis of how accurately they approximate Navier Stokes behavior at the macroscopic level. Based on this, the lattices are termed D2Q9, D3Q15, D3Q19, and D3Q27 respectively. The set of 9, 15, 19 or 27 directions are called lattice velocity vectors. They are depicted in Fig. 5.1.

As mentioned earlier, the method involves discretizing the Boltzmann behavior of a set of particles. The spatial discretization part of it is to make the particles move in a fixed set of directions only. To track the number of particles in a given direction on a particular location we use *particle distribution functions (PDFs)*. These are distribution function values of the particles and not their count, and hence can take fractional values. The particles are assumed to be unit mass. Each grid cell has equal sides measuring unity for ease in computation. At the beginning of the simulation the density of fluid within each grid cell is unity. This implies a

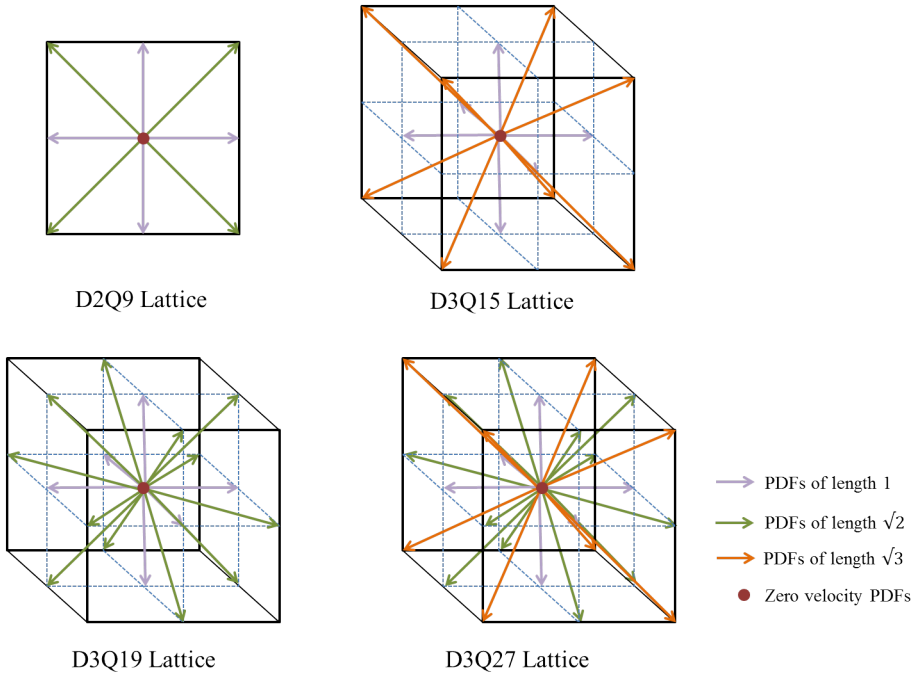

Figure 5.1: Various Lattices used for LBM simulations

Velocity Vector	Value
$\mathbf{e_1}$	$(0,0,0)'$
$\mathbf{e_{2,3}}$	$(\pm 1,0,0)'$
$\mathbf{e_{4,5}}$	$(0,\pm 1,0)'$
$\mathbf{e_{6,7}}$	$(0,0,\pm 1)'$
$\mathbf{e_{8\ldots11}}$	$(\pm 1,\pm 1,0)'$
$\mathbf{e_{12\ldots15}}$	$(0,\pm 1,\pm 1)'$
$\mathbf{e_{16\ldots19}}$	$(\pm 1,0,\pm 1)'$

Table 5.1: Velocity Vectors for D3Q19

single particle within each cell at $t = 0$. Also, since the sides of the cell are unit length and the particles are unit mass, at any point of time the density (ρ) is the number of particles within a cell (eq. 5.1). The velocity field element ($\mathbf{u}$) at any cell is the momentum density of the cell (eq. 5.2).

$$\rho = \sum df_i \tag{5.1}$$

$$\mathbf{u} = \frac{\sum e_i df_i}{\sum df_i} \tag{5.2}$$

5.1 The basic algorithm

Two steps comprise the basic algorithm, *collision* and *streaming*. A cell of D3Q19 lattice at $\langle x, y, z\rangle$ maintains a vector of 19 PDF values, $\langle df_0, df_1, \ldots, df_{18}\rangle$. *Streaming* involves the progression of PDFs in the direction of their corresponding velocity vectors. Hence, streaming is just a copy operation from $\langle x, y, z\rangle$ $[i]$ to $\langle x + e_{ix}, y + e_{iy}, z + e_{iz}\rangle$ $[i]$, $\forall i \in 0, \ldots, 18$. Post streaming, each cell has a new set of PDFs. Velocity and density for the current iteration are calculated using these. For each cell, *collision* involves calculating equilibrium distribution functions for each direction. The final vector of PDFs ($\langle df'_0, df'_1, \ldots, df'_{18}\rangle$) for the next iteration is arrived upon using the streamed PDFs and corresponding equilibrium distribution functions.

$$w_i = \begin{cases} \frac{1}{3} & \text{for } i = 1 \\ \frac{1}{18} & \text{for } i = 2...7 \\ \frac{1}{18} & \text{for } i = 8...19 \end{cases} \qquad \text{for filled cells}$$

$$df'_i = (1 - \omega)df_i + \omega df_i^{eq} \tag{5.3}$$

The complete process consisting of the two steps (collision and streaming) is represented by the equation,

$$df_i(\mathbf{x} + \mathbf{e_i}\Delta t, t + \Delta t) - df_i(\mathbf{x}, t) = -\frac{1}{\tau}(df_i(\mathbf{x}, t) - df_i^{eq}(\mathbf{x}, t)) \tag{5.4}$$

5.2 Simulation of free surfaces

The above method outlined the two basic steps for simulating the bulk of fluid. To simulate free surfaces (the partition between the fluid and the environment), we need to expand the algorithm to account for the interaction of the fluid with the environment. If something is completely surrounded by water, the flow can only be felt by it. However, if this is not the case

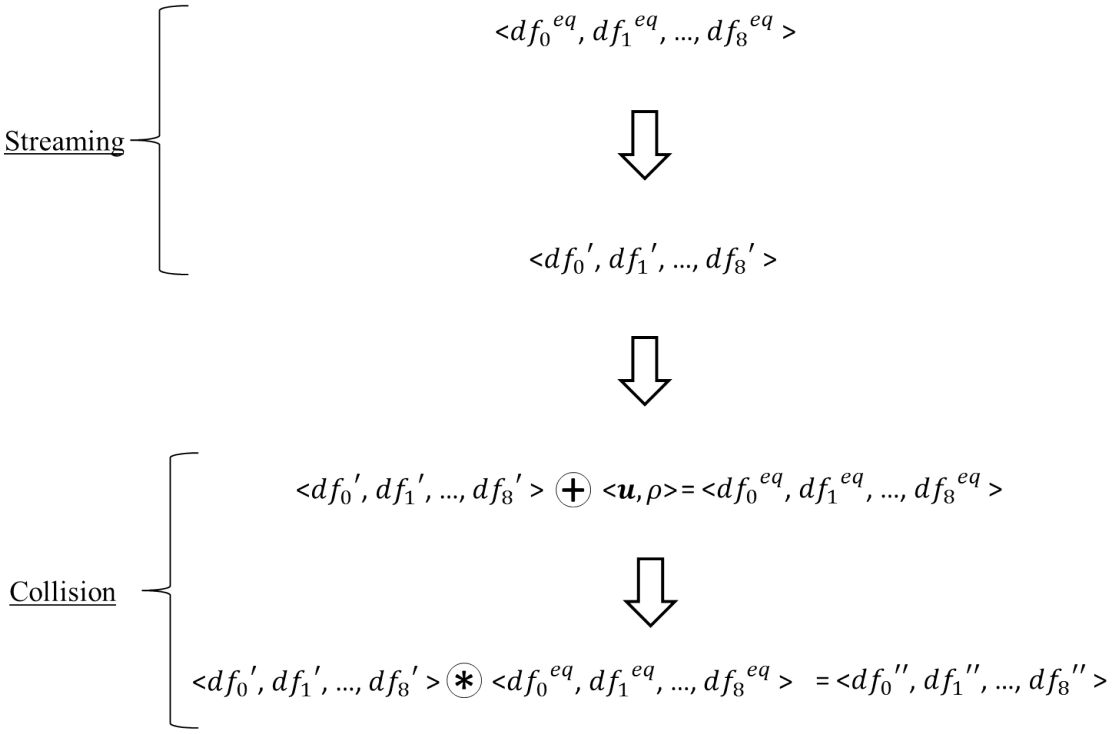

Figure 5.2: Basic LBM overview

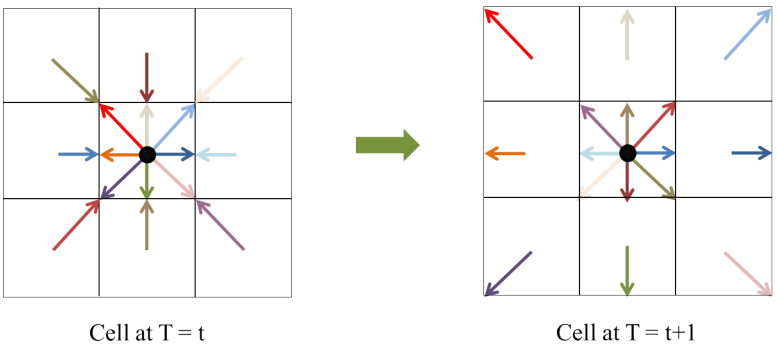

Figure 5.3: Streaming of Particle Distribution Functions

the fluid forms a separating boundary surface between itself and the environment is formed, known as free surface. The progress and shape of the free surface depends on many factors. The presence of external force, the shape of the domain, and fluid parameters (viscosity etc.) etc. Instead of beginning with each cell having a particle, we only put fluid particles in in cells where the fluid is present. These then travel from one cell to another, crossing into previously empty cells and progressively vacating previously filled ones. Thus we need to add to our algorithm to account for such changes.

5.2.1 Overview

The lattice used for the modified algorithm remains the same. However, the cells need to now be labeled according to whether they contain fluid, gas or both (partially). The partially filled cells will invariably be the separating layer between the filled cells and the empty/gas cells. As fluid progresses forward, we have interface cells gradually filling up with fluid up to the maximum capacity. They are completely filled and are labeled as such. The empty cells surrounding such cells are then marked as interface cells themselves - thus re-forming the partition between empty and filled cells. Since the system is isolated, progression of fluid to one place also means retreat of fluid from another. Thus we have cells emptying progressively and after being wholly empty are labeled as empty cells with the neighboring filled cells being relabeled as interface. The free surface is built out of the interface cells based on the amount of fluid they hold. This is given by the amount of fluid they hold, and by the magnitude of their PDFs in various directions. After each LBM iteration adjustments are made in the relaxation time for each cell based on the local stress tensor. Local stress tensors are calculated with the help of the current velocity field. Atmospheric pressure, reference density and pressure of fluid are assumed to be unity for simplicity.

5.2.2 Algorithm

Since the label on a cell depends on how much fluid it holds, fluid fraction ϵ is calculated for each cell. A fluid fraction is defined as ratio of the cell mass with its density.

$$\epsilon = \frac{m}{\rho} \tag{5.5}$$

This implies that we do not equate mass to density, like we did earlier, anymore. As PDFs are streamed, corresponding change in mass among cells also takes place. Naturally the change is equal and opposite between fluid cells. Also, there is no exchange of mass between an interface cell and an empty cell. However, there is a net mass exchange between an interface cell and a filled cell, given by,

$$\Delta m_i(\mathbf{x_i}, t + \Delta t) = df_{\tilde{i}}(\mathbf{x} + \mathbf{e_i}\Delta t, t) - df_i(\mathbf{x}, t) \tag{5.6}$$

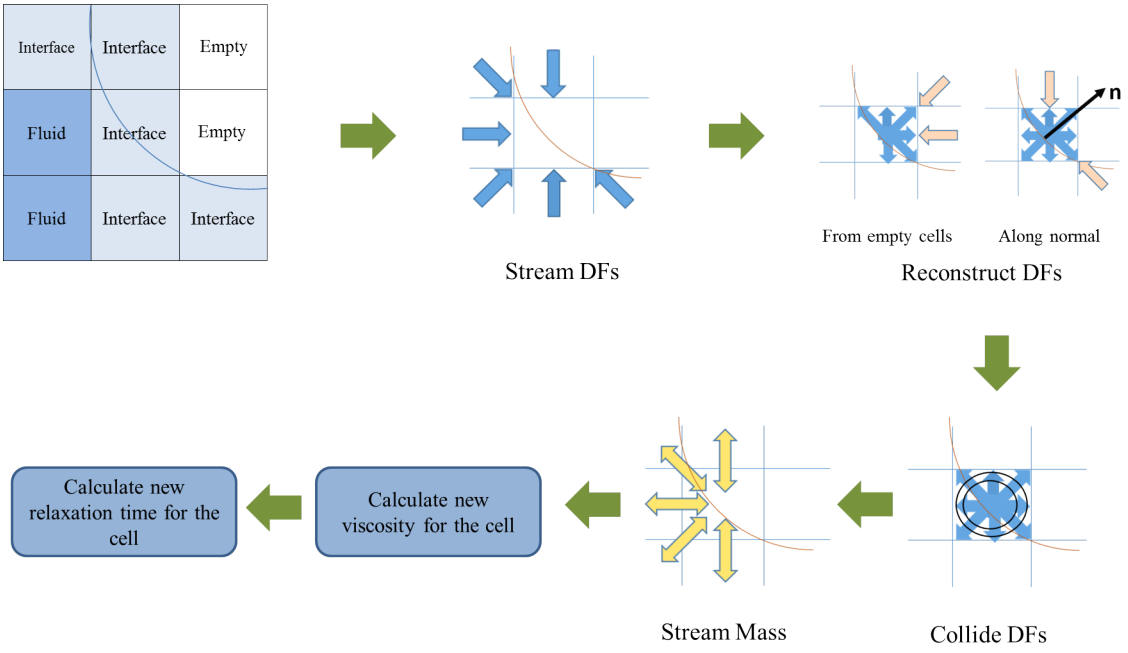

Figure 5.4: Algorithm for free surface simulation of Generalized Newtonian Fluids

Where $\tilde{i}$ is the direction opposite to i. So the first term in the R.H.S. is number (factional) of particles entering the cell and the second term is number leaving the cell. The mass transfer between two interface cells takes into account the area the area of fluid interface between the two. Thus,

$$\begin{aligned} \Delta m_i(\mathbf{x_i}, t+\Delta t) &= s_e \frac{\epsilon(\mathbf{x}+\mathbf{e_i}\Delta t, t)+\epsilon(\mathbf{x},t)}{2}, \\ s_e &= df_{\tilde{i}}(\mathbf{x}+\mathbf{e_i}\Delta t, t) - df_i(\mathbf{x},t) \end{aligned} \tag{5.7}$$

The cumulative mass exchange for a cell needs to be now reflected in the net mass for the cell. This is given by,

$$m(\mathbf{x}, t+\Delta t) = m(\mathbf{x},t) + \sum \Delta m_i(\mathbf{x}, t+\Delta t) \tag{5.8}$$

Next comes the streaming PDFs through the grid. As the gas cells are completely vacant they do not have any PDFs. Hence they don't participate in the streaming step. Streaming, both to and from the filled cells, is done in the regular fashion as described in the previous sections. Interface cells are handled differently, in that, if for a cell at $\mathbf{x}$,there is an empty cell at $\mathbf{x}+\mathbf{e_i}\Delta t$ then,

$$df_{\tilde{i}}' = df_i^{eq}(\rho_A, \mathbf{u}) + df_{\tilde{i}}^{eq}(\rho_A, \mathbf{u}) - df_i(\mathbf{x},t) \tag{5.9}$$

Where $\mathbf{u}$ is the velocity of the cell and ρ_A is the reference density of the fluid taken to be unity. Next, the PDFs coming from the direction of the interface normal are reconstructed using the

same equation. The interface normal is calculated as,

$$\begin{aligned} &\mathbf{n}.\mathbf{e_i} > 0 \\ &\mathbf{n} = \frac{1}{2}\begin{pmatrix} \epsilon(\mathbf{x_{j-1,k,l}}) - \epsilon(\mathbf{x_{j+1,k,l}}) \\ \epsilon(\mathbf{x_{j,k-1,l}}) - \epsilon(\mathbf{x_{j,k+1,l}}) \\ \epsilon(\mathbf{x_{j,k,l-1}}) - \epsilon(\mathbf{x_{j,k,l}} + \mathbf{1}) \end{pmatrix} \end{aligned} \tag{5.10}$$

Next, the standard collision step is performed for all filled and interface cells. The empty cells are ignored. Due to mass and PDF streaming, relabeling of the cells is now in order. If the (now) current mass density exceeds unity by more than a fixed amount (10^{-3}), the cell is labeled filled. Else if the cell's mass density is less than zero by the same amount, the cell is labeled emptied.

After the relabeling, we need to ensure that interface boundary filled and empty cells don't have holes in it. Also, the relabeled cells had mass density falling out of the permissible range. To ensure mass conservation we need to redistribute excess mass or remove deficient mass from the relabeled cells. We first calculate the neighborhood for the filled cells by ensuring empty cells, if any, being labeled as interface cells. Since these will not have any PDFs or mass of their own, an average of density and velocity of the neighboring cells is attributed to them. Their PDFs are initialized with equilibrium values obtained using this ρ_{Avg} and v_{Avg}. We also need to remove the interface cells which are needed for the boundary and which have been labeled empty in the last step by falling out of the range. Now, the flag of the filled cells is set and the emptied cells are labeled interface. The PDFs for the latter remain the same.

Next the m_{ex} of the filled cells is distributed to the neighboring interface cells. It is calculated as $m-\rho$. The m_{ex} is equal to the mass m (negative value) for emptied cells and is also distributed to the neighboring interface cell. The distribution is done according to the following eq. (5.11).

$$m(\mathbf{x} + \mathbf{e_i}\Delta t) = m(\mathbf{x}\mathbf{e_i}) + m_{ex}(\eta_i/\eta_{tot}) \tag{5.11}$$

Here,

$$\eta_{tot} = \sum \eta_i \tag{5.12}$$

$$\eta_i = \begin{cases} \mathbf{n}\cdot\mathbf{e_i} & \text{if } \mathbf{n}\cdot\mathbf{e_i} > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for filled cells}$$

$$\eta_i = \begin{cases} -\mathbf{n}\cdot\mathbf{e_i} & \text{if } \mathbf{n}\cdot\mathbf{e_i} < 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for emptied cells}$$

Along with mass change, the fluid fraction for cells also needs to be updated. Also, the order of filled cells being converted before empty cells is not important. We could have as well converted empty cells before filled ones. During interpolation therefore for empty cells, only those cells must be considered which aren't new interface cells (converted from the filled ones before) themselves.

We now have successfully updated PDFs for all fluid and interface cells. According to changes in relative velocity of neighboring cells, varying stress is introduced into the volume of the fluid. For a Newtonian fluid, this doesn't translate to a behavioral change. For a non-Newtonian fluid, however,this translates to a spatial variation in viscosity. The physics of non-Newtonian fluids has already been delved into in Ch. 2. Here, we describe how we incorporate the changes into the implementation. We use the truncated power law model to simulate non-Newtonian behavior.

For each fluid or interface cell, strain rate tensor is computed. In three dimensions, it is given by,

$$S = \begin{pmatrix} \frac{\partial \mathbf{u}}{\partial x} & \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial x}+\frac{\partial \mathbf{u}}{\partial y}\right) & \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial x}+\frac{\partial \mathbf{u}}{\partial z}\right) \\ \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial y}+\frac{\partial \mathbf{u}}{\partial x}\right) & \frac{\partial \mathbf{u}}{\partial y} & \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial y}+\frac{\partial \mathbf{u}}{\partial z}\right) \\ \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial z}+\frac{\partial \mathbf{u}}{\partial x}\right) & \frac{1}{2}\cdot\left(\frac{\partial \mathbf{u}}{\partial z}+\frac{\partial \mathbf{u}}{\partial y}\right) & \frac{\partial \mathbf{u}}{\partial z} \end{pmatrix} \tag{5.13}$$

The second invariant of stress tensor is then calculate,given by,

$$D_{II} = S_{0,0}\times S_{1,1} + S_{1,1}\times S_{2,2} + S_{2,2}\times S_{0,0} - (S_{0,1}\times S_{1,0} + S_{1,2}\times S_{2,1} + S_{1,3}\times S_{3,1}) \tag{5.14}$$

The shear rate γ is calculated as,

$$\dot{\gamma} = 2\times\sqrt{|D_{II}|} \tag{5.15}$$

The apparent viscosity for the cell is given by equation 5.16.

$$\nu = m\times\dot{\gamma}^{n-1} \tag{5.16}$$

Care should be given to the fact that the apparent viscosity value calculated here remain within the bounds in which LBM is stable. The relation between relaxation time and apparent viscosity is,

$$\tau = \frac{6\times\nu+1}{2} \tag{5.17}$$

As we know, τ is the reciprocal of relaxation frequency ω. Thus ω is calculated for each cell for the next time step. This *per-cell* relaxation frequency is used at collision time in the next instant.

5.3 Boundary Conditions

Boundary conditions are an essential element of any simulation. Accurate handling of interaction of the fluid with the spatial extremities of the domain is necessary. The walls of the domain provide an obstacle to the flow. The pressure exerted by the obstruction affects the flow of the fluid. Depending on the interaction of the fluid with the material of the boundary, we characterize the boundary. Take water, for example. The walls of the domain in contact with the fluid will either be hydrophilic, meaning attracting water, hydrophobic, meaning repelling water or anywhere between the two. There are ample surfaces in nature and around us which

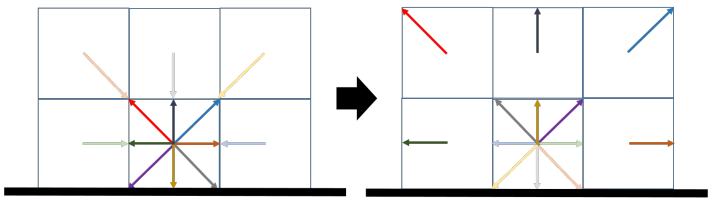

Figure 5.5: Bounceback boundary conditions

display such properties. For instance, the leaves of a lotus plant are almost perfectly hydrophobic. Water, slips off of them without leaving a droplet behind! The rocks made lime, on the other hand, due to their rough and porous nature, are extremely hydrophilic. The walls of a typical brick house, or our utensils lie somewhere in between. The hydrophilic or hydrophobic nature of the walls is simulated by imposing no-slip or free-slip boundary conditions.

5.3.1 No-slip boundary conditions

If we are to simulate walls which perfectly attract fluid, we basically need to impose a deterrent on the lamina in contact with the boundary so that the fluid there is perfectly stationary. For macroscopic top down methods of fluid simulation we need to ensure that the velocity field imposed on the fluid stays continuous and differentiable throughout the domain including at the boundary. This brings some inaccuracies in the methods as calculation of the derivative for the velocity field at the boundary is not applicable. An advantage of LBM over other methods is that it is easy to understand and code. To apply No-slip boundary conditions to an LBM simulation one has to remember that the fluid closest to the wall doesn't move. For an LBM cell, this means that the number of particles entering and exiting between two time instants is zero. Thus, the boundary just reflects PDFs of the incoming functions. Algorithmically, this means that the boundary adjacent cells have to be distinguished from the regular internal cells. The change in their behavior is reflected in the streaming step only. The PDFs coming from the regular cells are streamed normally. The PDFs to be streamed towards the boundary cells are reflected exactly. They remain in the same cell with just their directions reversed. Hence, this is also known as bounce-back boundary conditions. This is shown by eq. (5.18) where $\tilde{i}$ is the direction opposite to i.

$$df_i(\mathbf{x}, t + \delta t)' = df_{\tilde{i}}(\mathbf{x}, t) \tag{5.18}$$

5.3.2 Free-slip boundary conditions

Free-slip boundary conditions as the name suggests, leave the fluid free to slip with zero friction on the obstacle walls. This behavior is shown by hydrophobic substances, such as extremely smooth glass surfaces etc. The velocity tangential to the boundary surface remains preserved

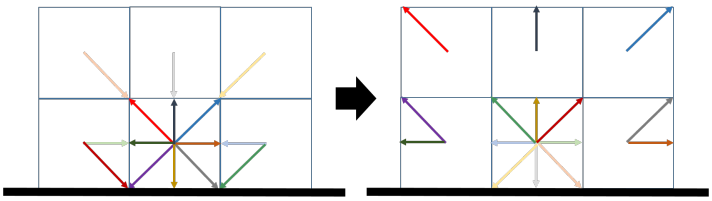

Figure 5.6: Free-slip boundary conditions

even as the velocity of the fluid normal to the boundary vanishes. As with the bounce back scheme, the obstacle cells are marked. Here however, the preservation of transverse/normal velocity requires the PDFs of the neighboring cell while streaming, unlike bounce back scheme where streaming was treated locally. Special cases arise when the neighboring cell in the direction of tangential velocity is not a fluid cell, for example, at corners. For such cells, bounce back scheme is used.

5.3.3 Real World Surfaces

More often than not, in the real world, surfaces, are neither purely hydrophilic not hydrophobic. Hence while simulating flows on these surfaces neither bounce back scheme nor no-slip scheme yields accurate results and a linear interpolation between the two conditions is used.

$$df_i(\mathbf{x}, t + \Delta t)' = \alpha df_{\bar{i}}(\mathbf{x}, t) + (1 - \alpha) df_r(\mathbf{x}, t) \tag{5.19}$$

α is the interpolation factor in eq. (5.19), with $\alpha = 0$ signifying free-slip and $\alpha = 1$, bounce-back boundary conditions respectively. df_r signifies the reflected PDF used in free-slip conditions. Here also, special cases such as corner boundary cells are handled separately suing pure bounce back scheme.

5.3.4 Neumann Boundary conditions

Often fluid flow simulations are constrained within boundaries which are not orthogonal or parallel to one another. In such cases merely modifying the net velocity vector uniformly of all the cells after an LBM call is not sufficient. There are other interactions which may be lost in this generalization and sometimes the simulation may not even work remotely similar to what is expected! Examples may make this problem clear. The problem is simulating the flow of a fluid, say, water in a U-shaped horizontal tube, as shown in the Fig. 5.7. The gravitational force acts vertically and water is being pumped through the tube, with fluid going in through the lower end and coming out of the upper end. The water inside, by virtue of being pumped flows with constant velocity. Hence, we cannot just uniformly change the velocity along particular direction for each cell after every LBM call. Consider another flow problem. Fluid is flowing in a horizontal Y-shaped tube. After reaching the split in the tube, how do we calculate what

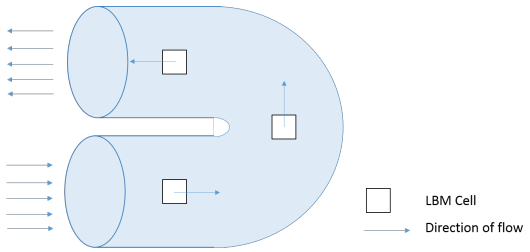

Figure 5.7: Fluid flow through a curved tube

modifications we need make to the velocities of each cell? The velocity field would be continuous and, in each arm would be parallel to the walls. When would the field lines start to divert in the main arm and how sharply would they divert? We cannot use naïve methods such as velocity modification at the local cell level according to its position in the domain. That will be brute force and not optimal. In such cases we make use of the Neumann boundary conditions. Neumann boundary condition involves prescribing the gradient of the variable field normal to the boundary.

$$\partial_n \mathbf{u}(\mathbf{x}) = constant \tag{5.20}$$

For our tube examples we prescribe the velocity of the fluid at the end points of the tube. We assume smooth gradient between the inlet and outlet. Applying regular (no-slip) boundary conditions on the walls of the tube we carry the simulation forward. A detailed description of Neumann boundary condition is given in Chen *et al.* [28].

5.4 Conclusion

In this chapter we detailed the algorithm for free surface LBM simulation. Basically, for single phase simulation, it comprises only a streaming step and a collision step. However, for multi-phase flow we need to take care of the interface between two fluids. We took the second fluid here to be air (with unit constant density) and extended the algorithm to take care of the interface by essentially computing the mass transfer - due to the progress of the interface after each iteration - to its neighbouring cells. This distribution is adjusted by converting the state of the cells if they're emptied or partially or completely filled. Moreover, the normals computed here can be stored and later used for lighting computations. We may use any of the three boundary conditions (no-slip, free-slip, or, Neumann) in our simulations. We show these in Ch. 7. To round up this chapter, we provide a pseudocode implentation of the algorithm, for better understanding.

Algorithm 1 Free Surface LBM for D3QY lattice

```
1: procedure reconstructDF(x, y, z)
2:     Update df using Eq 5.4
3: end procedure
4: procedure transferMass(x, y, z)
5:     Update mass using Eq 5.6
6:     if cell becomes completely filled then
7:         Mark as filled_interface_cell
8:     else if cell becomes completely empty
9:         Mark as emptied_interface_cell
10:    end if
11: end procedure
12: procedure relabelCells(x, y, z)
13:    if filled_interface_cell then
14:        Convert empty neighbours into interface cells
15:        Make current cell a fluid cell
16:    else if emptied_interface_cell
17:        Convert fluid neighbours into interface cells
18:        Make current cell an empty cell
19:    end if
20: end procedure
21: procedure distributeExcessMass(x, y, z)
22:    if filled_interface_cell or emptied_interface_cell then
23:        Distribute excess mass among neighbours
24:    end if
25: end procedure
26: procedure calculateNewViscosity(x, y, z)
27:    Calculate viscosity using truncated power law
28: end procedure
29: procedure Free Surface LBM
30:    for all cells in parallel do
31:        if fluid or interface cell then
32:            stream(x,y,z)                          ▷ Same as Basic LBM
33:        end if
34:        if interface cell then
35:            reconstructDF(x, y, z)
36:        end if
37:        if fluid or interface cell then
38:            collide(x,y,z)                         ▷ Same as Basic LBM
39:        end if
40:        if interface cell then
41:            transferMass(x, y, z)
42:            relabelCells(x, y, z)
43:            distributeExcessMass(x, y, z)
44:        end if
45:        if Non Newtonian fluid then
46:            calculateNewViscosity(x, y, z)
47:        end if
48:    end for
49: end procedure
```

Chapter 6

Lattice Boltzmann Method on a GPU

In Ch. 5, we gave detailed the algorithm for implementing LBM for GNFs on the computer, without using a GPU. While the algorithm is easy to code, due to the massive computational requirement, a serial CPU implementation is not sufficient to provide interactivity or speed to simulation while maintaining an acceptable level of detail. To counter the problem, we therefore look at parallel methods. In this chapter we detail our GPU implementation for LBM. We explain the arrangement of data and the multithreading technique used by us to make the algorithm suited for the GPU.

LBM is inherently suited for parallelization. Each cell of the lattice, at every time instant, undergoes the same operations. This is followed by data transfer. Further, some steps of the algorithm can be executed independently from one another, and the results synced. This fits well with the data parallel model of parallelization of the GPU. We also implement the algorithm on multiple GPUs to further parallelise the process. The following sections show how we implement the algorithm on a GPU.

6.1 Data Requirement

Analyzing the data requirement of our simulation is important for its efficient storage and transfer from the memory. The data requirement is provided in the following Table 6.1. The distribution functions are stored in two different sets of values, one previous and one current for double buffering. For each iteration, we track the state of the cells as they need differential treatment according to whether they are filled, empty or interface. The states would be updated post an iteration according to how much fluid each cell holds. We also need to track the mass density and velocity for each cell every iteration. These are computed and consumed within the iteration, thus not requiring a copy after each timestep.

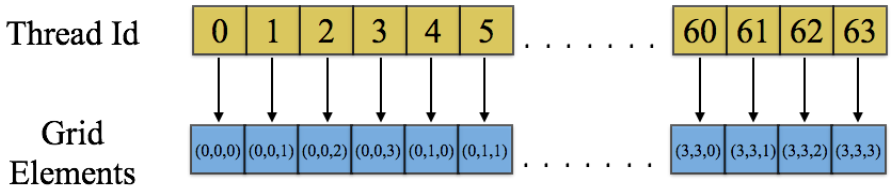

Figure 6.1: Thread Mapping with Grid Elements

6.2 Thread Mapping

Since each cell reads its neighbour's previous data and writes only its own current data, the computation for each cell happens independent of the others. Thus, we assign one thread per cell for doing the computation.

We flatten the grid structure to make a 1D grid of threads and map each thread to the grid elements in row major order as shown in Figure 6.1.

Because each warp consists of 32 threads, for grid sizes with an x-dimension multiple of 32, each warp operates on cells which lie in the same row, thus leading to an optimised access as explained in the following sections.

6.2.1 Data Layout

For efficiency, it is critical to store the data in a manner which allows maximum possible coalesced read and write operations. To achieve this, we employ a *SoA* (Structure of Arrays) data format to store the information required for each cell, wherein the data for the 3D grid is stored linearly in the memory as a 1D array in row major order.

The distribution function is stored the same way, with the values corresponding to a particular direction stored in contiguous memory blocks in row major order, as shown in Figure 6.2.

Data	**Size**	**Use**
Previous DFs	19 floats	Previous iteration distribution function
Current DFs	19 floats	Current iteration distribution function
Previous State	1 int	Type of cell in previous iteration
Current State	1 int	Type of cell in current iteration
Epsilon	1 float	Intermediate, visualisation purposes
Velocity	3 floats	Intermediate, visualisation purposes

Table 6.1: Data Requirement for each cell

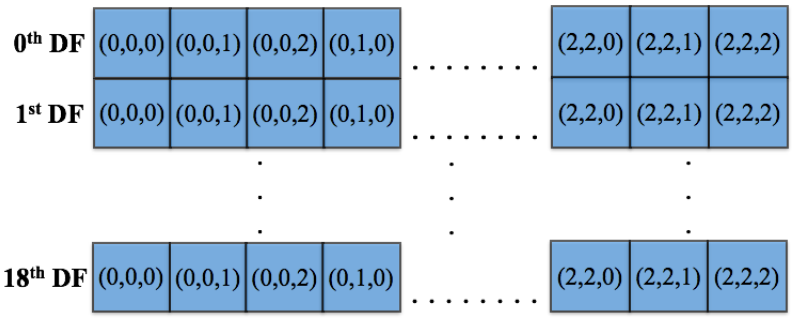

Figure 6.2: Distribution Function Layout for a 3^3 Grid, stored in row major format

6.2.2 Memory Access Pattern

In `stream`, `reconstructDF` and `collide` kernels given in Algorithm 1 (Chapter 4), all threads in a warp update the distribution function for a particular direction at the same time. In the `transferMass` kernel, the distribution function is read in the same manner. These memory accesses are fully coalesced because adjacent threads map to horizontally adjacent cells of the grid. For instance, if a thread with thread index (tid) maps to the cell $\langle x, y, z\rangle$, then the thread $(tid+1)$ will map to the cell $\langle x+1, y, z\rangle$. Their k^{th} neighbour would be $\langle x+e_{ix}, y+e_{iy}, z+e_{iz}\rangle$ and $\langle x+1+e_{ix}, y+e_{iy}, z+e_{iz}\rangle$ respectively, where $\langle e_{ix}, e_{iy}, e_{iz}\rangle$ is the k^{th} direction vector. Hence, the k^{th} neighbour of adjacent cells are also adjacent. Because of the SoA data layout, the distribution function values of a particular direction for the k^{th} neighbour of adjacent cells are also adjacent in memory. This is shown in Figure 6.3. All above kernels achieve 100% occupancy on the GPU hardware.

The remaining steps, `relabelCells` and `distributeExcessMass`, read their neighbour's data and update their own. Since neighbours of adjacent cells are adjacent in memory, these too are coalesced accesses. These kernels only achieve 75% occupancy of the GPU due to the need for more registers to hold the variables used.

6.2.3 Thread Divergence

The steps for Free Surface LBM are performed only for the interface cells. Only the interface cells partake in generation of the free surface, the filled fluid cells essentially undergo only the basic LBM operations of streaming and collision (the mass transfer whcih happens between two filled cells is equal and opposite). Therefore, the kernels are called for all cells, but will generally modify only the interface cells. This introduces thread divergence in the kernels. One solution to avoid this is to sort the cells according to their state. However, the memory locations which the threads work on would still be disparately arranged. Because of this, adjacent threads don't work on adjacent cells in memory, thus leading to uncoalesced memory accesses. To achieve coalesced memory access, data also needs to be sorted accordingly, which

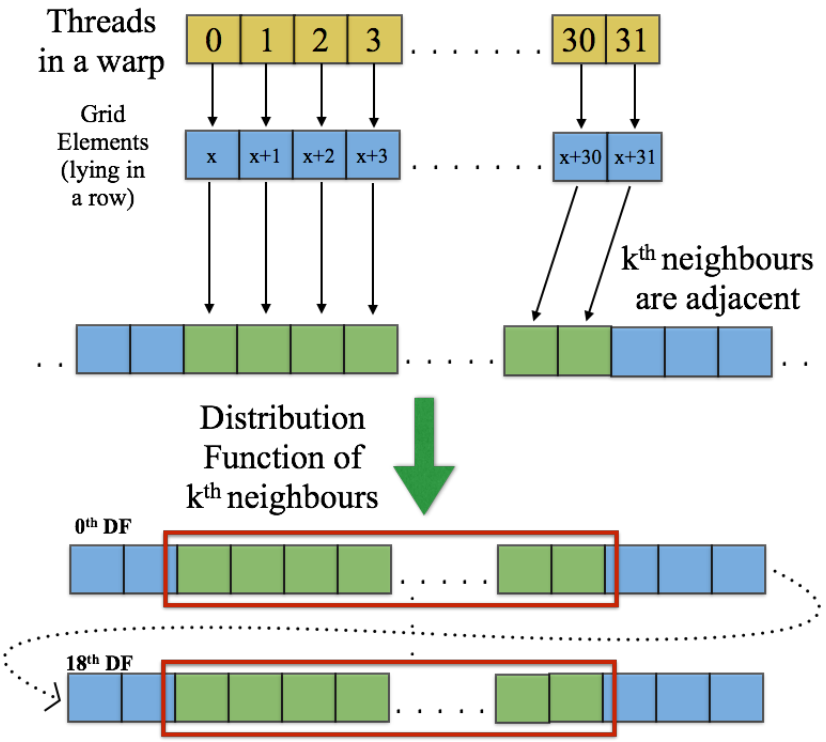

Figure 6.3: DFs for k^{th} neighbours of adjacent cells

is computationally expensive and worsens the situation. Thus this way the problem is solved but at heavy computational cost making the process much slower.

On the other hand, the interface cells form the boundary of the liquid and are much less in number. The threads corresponding to the non-interface cells simply return and there is thread divergence only for those warps which have both interface and non-interface cells. Thus, the overhead of thread divergence is much lower than the computational overhead of separating the interface cells and running it only for them.

6.3 Multi-GPU Implementation

We use two GPUs on the same system to further scale the problem. We divide the data for each GPU by slicing the grid along the z-axis. We do not choose the y-axis because bulk of the fluid is present at the bottom of the grid, which would lead to uneven distribution of the filled and interface cells among the two GPUs. The x-axis is not chosen to exploit the spatial locality along it. The cells on the boundary of the dividing slice need the data from the neighbouring cells which reside on the other GPU. So, in each iteration, the slice of data on the boundary is transferred to the other GPU. Each GPU needs to transfer the current DFs and state of the boundary cells to the other GPU. As evident from the pipeline shown in 6.4, the DFs are

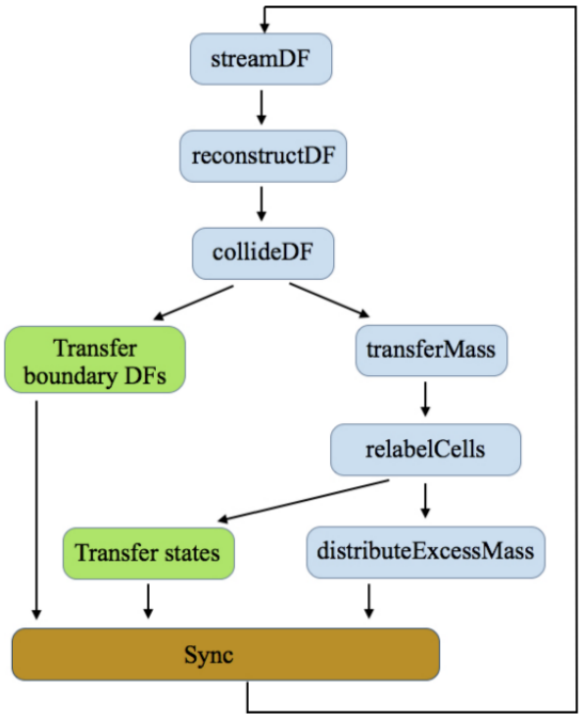

Figure 6.4: Overlap with data transfer with computation

available as soon as collision step is completed and are not required until the next iteration. So, we do an asynchronous transfer to the other GPU to overlap it with the computation. Similarly, the states are transferred as soon as they are reinitialised.

6.4 Conclusion

In the preceding sections we gave a way to parallelise free surface LBM simulations on a GPU. We assessed our data requirements fro effective implementation. We utilise *Structure of Arrays* to arrange the data and go for efficient thread mapping that provides for most of the data transfer to happen to contiguous locations, thus providing for an optimum memory access. In case of interface cell computations, we do not restrict our kernels to operate on only those cells which form the interface, resulting in thread divergence. This is an undesired result of our implementation, however, as shown above, its overhead is lower in comparison to the potential overhead cherry picking the interface cells to have the kernel run on. All these lead to significant performance enhancement, which shall be showed in Chapter 7.

Chapter 7

Results

Any fluid solver needs to be judged on two aspects, qualitative and quantitative. The quantitative parameters look at the accuracy with which it reproduces the fluid behavior i.e. how similar are the results with the analytic experiments or the actual observed parameter values. The qualitative parameter would be visual accuracy - how *real* does the simulation look. While the former is necessary for scientific purposes, the latter is given more emphasis by the animation/gaming industry.

We designed four experiments to challenge the solver both qualitatively and quantitatively. These are,

- Flow between two parallel plates.
- Lid driven cavity experiment.
- Dam Break Experiment.
- Flow of a non-Newtonian fluid through a tube of varying cros-section.

The first section describes experiments done on a CPU while the second gives the performance analysis of the same experiments on a GPU.

7.1 Experiments conducted on a CPU

7.1.1 Flow between two parallel plates

The first experiment was conducted for two dimensional flow. The aim was to look at the velocity profile for a liquid as it is made to pass between two parallel plates. It is assumed that the plates have a large area so that the fluid flows just between them and not around. A motion parallel to the two plates is induced in the fluid. The fluid lamina in contact with the two plates will not move on account of its viscosity. As we move further away from either of the two plates the velocity of each fluid lamina increases, until we reach the center, where,

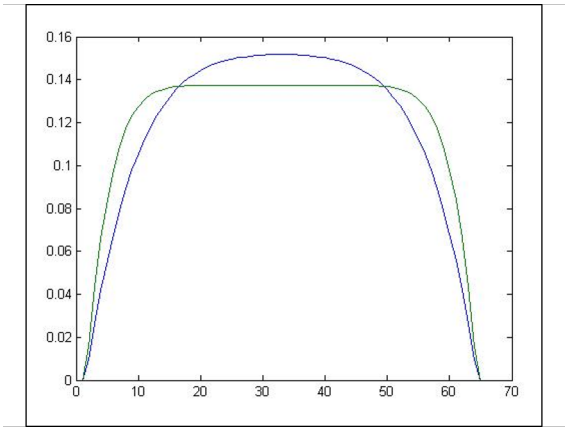

Figure 7.1: Comparison between flow curves of Newtonian (blue) and non-Newtonian (green) Fluids

due to symmetry the velocity is expected to be the maximum. Analytical calculations augur a parabolic velocity profile for a Newtonian fluid. For a non-Newtonian fluid, the profile will be more complex since shear between laminae will give rise to changes in viscosity. These changes in viscosity correspondingly would affect the velocity of the laminae. Indeed, for a pseudo-plastic fluid, it has been shown that that a parabolic curve which is plateaued (flattened) in the center is to be expected. Fig. 7.1 shows the normalized velocity profiles obtained from our experiment. The channel width was kept to eighty units. The blue curve is for a Newtonian fluid whereas the green curve is for a pseudo-plastic fluid. It can be seen that where as the blue curve follows a parabolic path the green curve flattens on approaching the center of the channel. The experimental results therefore, conform to the analytical expectation.

7.1.2 Lid driven cavity experiment

This experiment is a standard problem for two dimensional fluid solver. The fluid is present in a rectangular domain, one of whose side is moving. In our set-up the side at the top is moving with a constant velocity. The boundary condition used here is *no-slip*. Instead of a regular fluid we have a non-Newtonian (pseudo-plastic) fluid inside the domain. After some time, vortexes appear, as expected. We track a vortex using a virtual dye of particles. The simulation begins with some particles per voxel. These particles are mass-less (logical) and after each collision-stream cycle are displaced according to the velocity of each cell according to bi-linear interpolation and their new positions are stored. The motion of the top side of the domain is dissembled by marking a set of cells as *accelerator cells*. The velocity of these cells is set to a fixed value in the x-direction and the density to one. The distribution functions are then computed around their respective equilibria as usual. Typically, the layer of grid cells right next to the obstacle cells layer is taken as the accelerator cell layer. A vortex formation suggests a movement of fluid around the vortex center. Hence, for a non-Newtonian fluid, we

expect a change in viscosity as come from the sides to the center of the vortex. Results for shear thinning fluid are shown in Fig. 7.2. The formation of vortex can be seen in successive frames. The change in viscosity is depicted by the color gradient. Red shows the area with maximum viscosity followed by blue and green respectively.

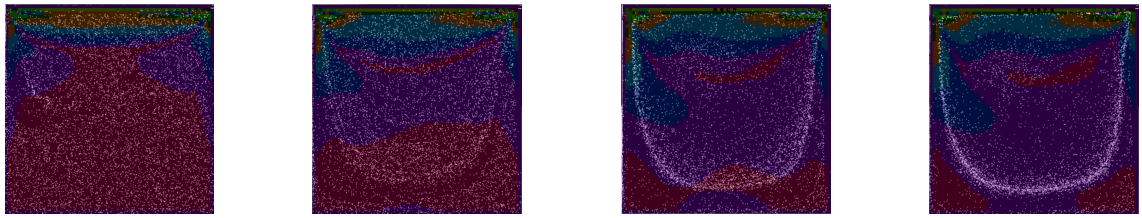

Figure 7.2: Lid-driven Cavity Experiment for a Shear Thinning Fluid

7.1.3 Dam Break Experiment

This is another one of the standard experiments which a fluid solver is subjected to. In a dam break experiment a finite volume of fluid is released in a cuboid shaped compartment. Initially, i.e. before the start of the simulation the flow of the fluid is kept in check lock-gates. The simulation starts with the removal of the gates/obstructions leading to the creation of a gravity current in the fluid. We performed this experiment for both Newtonian and non-Newtonian fluids. For Newtonian fluids, we used free-slip, no-slip and intermediate boundary conditions were used. For non-Newtonian fluids only no-slip conditions could be used for obvious reasons.

7.1.3.1 Dam Break with a Newtonian Fluid

This provides a quantitative assessment of the solver. Things to note is the visual accuracy of the solver, the formation of waves, symmetry and interaction of the fluid with solid obstacles and interaction of distinct volumes of fluid upon collision. Fig. 7.3 show the results obtained.

7.1.3.2 Dam-Break with a non-Newtonian Fluid

This proves as both a qualitative and quantitative check of the solver. The fluid, upon the removal of the stop-gates experiences gravity due to its bulk. The field leads it to spread and assume the shape of the container. The shear resulting from the interaction of fluid lamina with the container walls leads different viscosities generated in the fluid. In the figures below we slice the simulation in half and show one half so that the changes both in the bulk as well as on the surface can be observed. The parameters were set to simulate a shear-thinning fluid hence the presence of shear rate gives rise to a drop in viscosity. The gradient shows the change in viscosity with red being the maximum/original viscosity and blue the least.

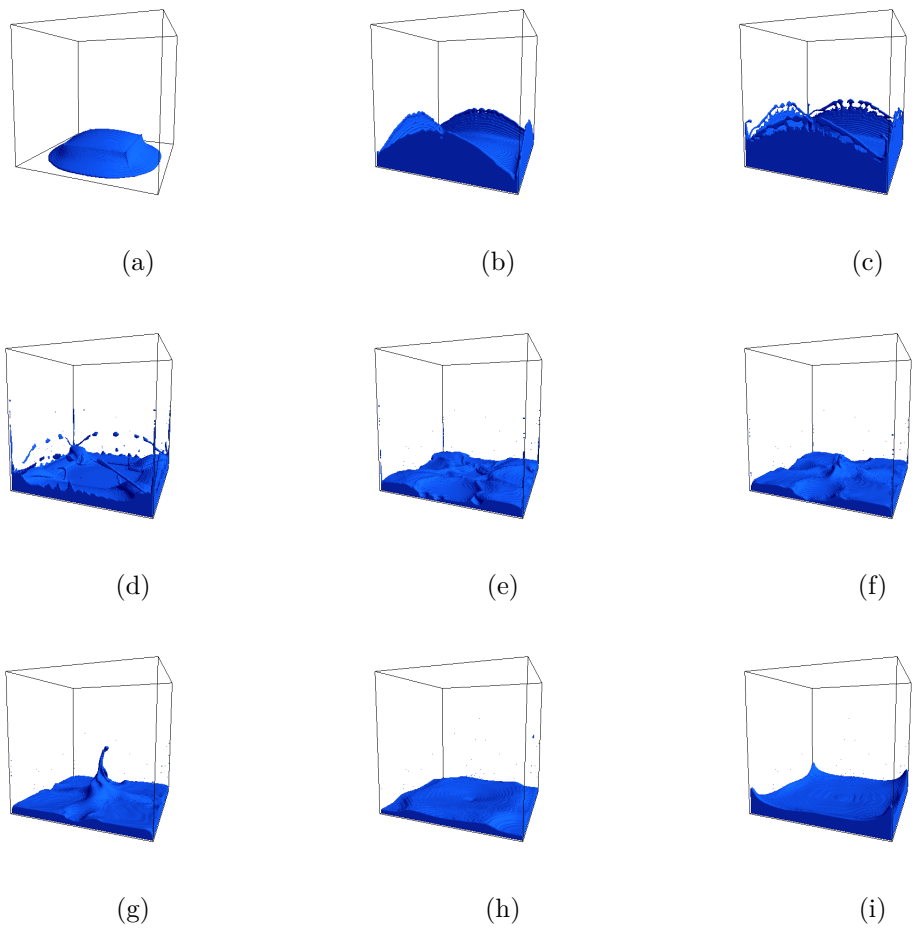

Figure 7.3: Dam Break Experiment for a Newtonian Fluid

7.1.4 Flow of a non-Newtonian fluid through a tube of varying crossection

We simulate non-Newtonian flow through a variable crossection tube. In order for a flow to be maintained we need something to drive it. The usual trick of tweaking the component of velocity parallel to the direction of applied force for each cell after each time step may not work here. This is because varying crossection of the tube as well as its probable twists and turns would make fluid particles move along stream lines which may not always conform to the direction of the applied force. Consider, for instance, a U-shaped tube with the start and terminating ends on the same side. In this extreme case, we can't tweak the cell velocities after every time step. That would result in zero flow with pressure applied at both ends. The situation is depicted in the following figure.

Therefore we apply Neumann boundary condition to the simulation. This involves having no-slip conditions on the side boundaries and maintaining a pressure difference between the two ends of the tube.

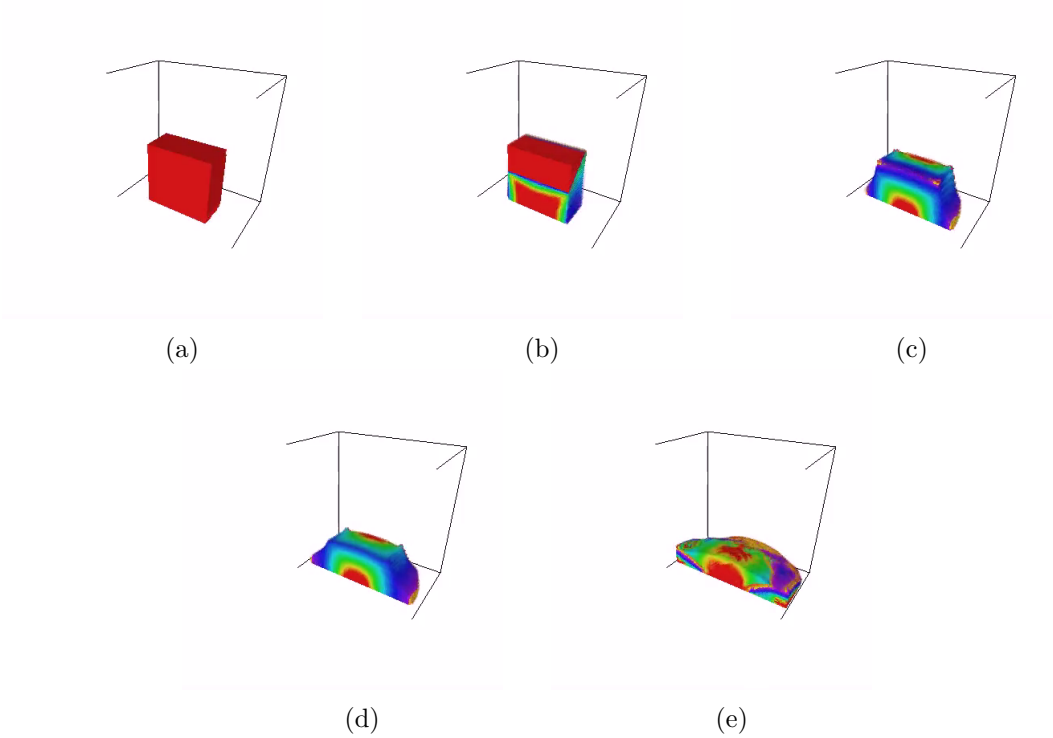

Figure 7.4: Dam Break Experiment for a Shear Thinning Fluid

Fig. 7.5 shows the flow simulation of a shear-thinning fluid through a tube having varying crossection. The flow is tracked by virtual dye with dye particles changing colors according to the specific viscosity areas they lie in. The start of the simulation has the bulk of the fluid as uniformly viscous. As time progresses the different regions are formed concentric with the tube with the inner most (having unhindered flow) retaining the original viscosity. The viscosity then progressively decreases.

7.2 Experiments on the GPU

In the previous sections, our focus was on testing the scientific accuracy of the method as well as testing the realism obtained in the simulations. In our GPU experiments we test the latter, along with testing the efficiency of the implementation. The output are standard simulations of dam break, falling drop, flow of Newtonian and non-Newtonian fluids through a slit etc. The experiments discussed below are performed on the NVIDIA Tesla K20c, unless stated otherwise.

The performance for the dam break experiment on various GPUs and grid sizes is given in the Fig. 7.6. The performance is measured in *Million Lattice Updates Per Second* (MLUPS), which is the number of grid points processed per second. The optimal block size for all the

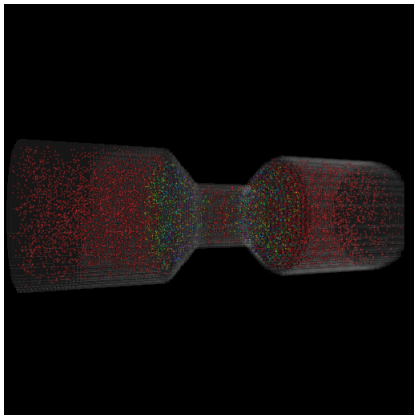

Figure 7.5: Flow of a shear thinning fluid through a tube with varying crossection

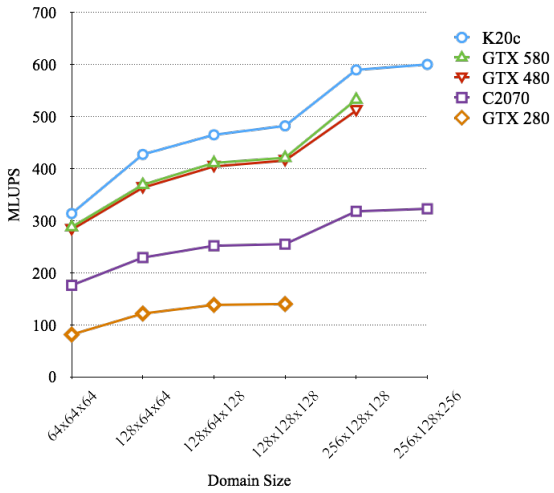

Figure 7.6: Performance of the Dam Break Experiment on various GPUs

GPUs is experimentally found to be 256, except for NVIDIA GeForce GTX 280, for which, it is 128. At these block sizes, the blocks fill up the GPU, giving close to 100% occupancy on most kernels.

Fig. 7.7 shows relative percentage of time taken by each kernel for 1000 LBM iterations of the dam break experiment on a 128^3 grid. As expected, the `collide` step takes the most amount of time because it is run for both *filled* and *interface* cells and updates their DFs after computing $\mathbf{u}$ and ρ.

7.2.1 Multi GPU Experiments

We performed the same experiments on two NVIDIA Tesla K20 GPUs. We follow the procedure described in the Ch. 6 for multiple GPUs. The results for dam break experiment are displayed graphically in Fig. 7.8. We see that for small grid sizes, the performance of

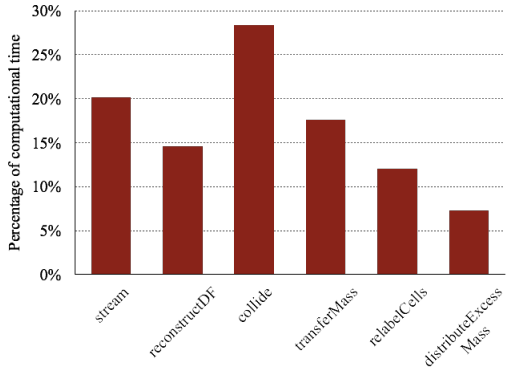

Figure 7.7: Relative time taken by each kernel on K20c for Dam Break Experiment on a 128^3 grid

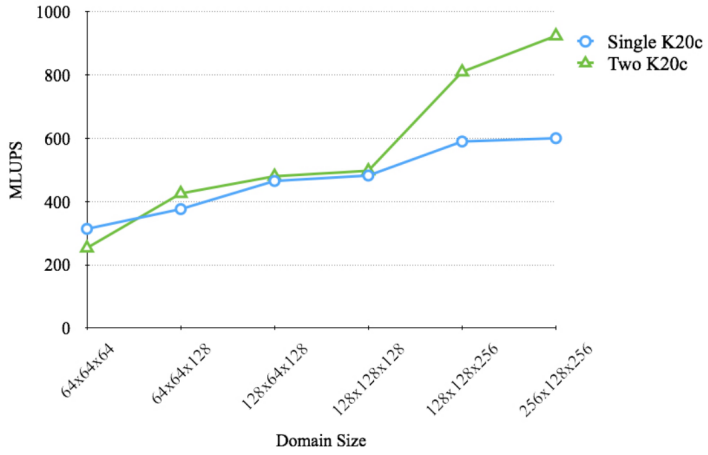

Figure 7.8: Comparative results for Dam Break Experiment on single and two GPUs

the two methods is comparable. However, for larger grids there is a considerable enhancement in performance in multi-GPUs. This is because for large grid sized, both the GPUs are well occupied.

7.3 Conclusion

In this chapter, we showcased the results obtained from our implementation. We showed that the algorithm gives physically real simulations. We proved the accuracy by the velocity profile in parallel plate experiment and supplemented it with lid-driven cavity and dam break experiments for non-Newtonian fluids which were color coded according to viscosity variations. We also showed that our GPU implementations are efficient and give interactive simulations whose performance increases with the increase in grid sizes when using more than one GPUs.

Chapter 8

Conclusion and Future Work

Through this work we have presented a method we were fascinated by, Lattice Boltzmann Method. Extremely intuitive and easy to implement, it does not compromise on accuracy. We depicted how starting from a particular lattice structure (D3Q19) it conforms to Navier Stokes' equation at the macroscopic level. The collision operator given by the BGK approximation make it extremely easy to code.

LBM is different from other simulation methods because, it is based on physical conservation laws. The conventional methods (Eulerian, Lagrangian or Finite Element methods) are based more on applying discrete mathematics techniques to a set of partial derivative equations (the Navier Stokes' equations). Thus they require more theory than the other methods. The methodology is based in statistical mechanics, unlike finite differences or spectral methods which are more mathematical applications. Also, the method is fascinatingly symmetrical. This chiral symmetry is not only aesthetic but is extremely important in similarity of the simulation with actuality. Spurious invariants need to be detected and pruned.

This is not to say that statistical methods do not have disadvantages. Some of them are, how to construct models for differential equations, statistical noise (more of a problem for LGCA than LBM), stability of LBM, large memory requirement of LBM. In contrast, the advantages include low complexity of code, flexibility with respect to domain geometry, increased accuracy in its range of stability overconventional methods, adaptability to parallelization.

8.1 Future Work

We encountered memory constraints when implementing the method on the GPU. The data in our implementation resided in the global memory of the GPU. Improving on the algorithm to utilise the shared memory space would lead to a truly optimal parallel implementation. We also encountered the issue of thread divergence while trying to cut down on needless computation cycles by only having the algorithm run on the required cells (as a considerable number of cells at each point in time would by empty on which no computations are required). Also, as one

goes for greater realism, lighting the simulation cannot be ignored. this is a computationally expensive process in itself and thus the need is to couple LBM with ray-tracing on the GPU and maintain real time performance. Porting the method on to smaller hand held devices for use in mobile gaming is also an interesting area of exploration. These are certain areas where future work should be directed.

Another area which could be explored is medical application of application of LBM. Fluids such as blood are non-Newtonian in nature. However, since the size of the blood vessels matters. As the vessels become thinner and thinner the non-Newtonian nature becomes more and more prominent. These factors need to be studies in order to make the algorithm more accurate at such scales. Another interesting area of development is turbulent fluids. LBM is constrained by a range in which it is stable. It would be interesting to see how far the range can be pushed and if in that range turbulent behavior can be simulated by it.

Chapter 9

Appendix

9.1 Visulisation

We detail the methods we used to visualize the LBM implementation for generalized Newtonian fluids. For generating free surfaces, we used Marching Cubes. While visualizing flows without a free surface, we use virtual dye particles. All simulations are done in openGL.

9.1.1 Marching Cubes

Marching cubes algorithm is a volume visualization algorithm which was introduced in 1987 by Lorensen and Cline for medical imaging applications. It created triangle models of constant density surfaces from 3D medical data. The algorithm assumes linear change between two neighboring values on the grid. The algorithm essentially, *marches through the grid*, one cube (comprising eight grid points) at a time, and replaces each cube with a polygon. The eventual mesh is the fusion of all the polygons. The algorithm was chosen for its simplicity in form and implementation without compromising on accuracy. It follows the simple formula of finding an isosurface point having a value of m in between adjacent points of a cube having values $m1$ and $m2$ respectively. Hence for a point $\vec{x}$ between two points $\vec{p_1}$ and $\vec{p_2}$,

$$\vec{x} \;=\; \frac{m_2 - m}{m_2 - m_1}\vec{p_1} + \frac{m - m_1}{m_2 - m_1}\vec{p_2}. \tag{9.1}$$

After obtaining a point $\vec{x}$ for each of the edges of the cube the algorithm creates a polygon out of them and progresses forward. The creation of a polygon is done by looking up in a table having $2^8 = 256$ pre-calculated polygon configurations. Each of the eight nodes of the cube are represented by a bit. If the field value at a node is greater than isosurface value, the appropriate bit is set else it is reset. The following figure shows the originally published fifteen cube configurations.

We fix the scalar field (mass-density) value for the isosurface at 0.5

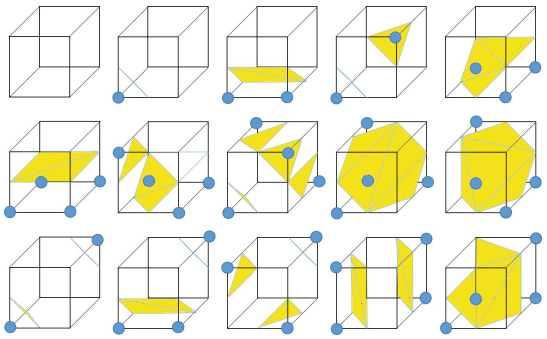

Figure 9.1: Original 15 marching cubes configurations for isosurface generation

9.1.2 Virtual Dye

It becomes difficult to track the flow of fluid where, although bounded rigidly, it does not interact with a substance with flexible contours or position which is affected by its interaction with the fluid. Therefore in cases like flow of a fluid in a tube we have to use marker particles for tracking. These particles serve the purpose of a virtual dye injected in the stream. Initially, we set the number of dye particles. Each particle is randomly awarded position in the domain to ensures a uniform distribution of particles within the domain at the start of the simulation. Then, after each time step, the position of each particle is interpolated (trilinear interpolation) according to the velocity of cell and its neighbours.

Related Publications

1. Tripathi, N. and Narayanan, P.J. *Generalized newtonian fluid simulations.* 2013 Fourth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG-2013). Jodhpur, India.

2. Jain, Somay and Tripathi, Nitish and Narayanan, P. J. *Interactive Simulation of Generalised Newtonian Fluids Using GPUs.* Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing (ICVGIP-2014). Bangalore, India.

Bibliography

[1] N. Foster and D. N. Metaxas, "Realistic animation of liquids," in *Proceedings of the Graphics Interface 1996 Conference, May 22-24, 1996, Toronto, Ontario, Canada*, 1996, pp. 204–212.

[2] J. Stam, "Stable fluids," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128. [Online]. Available: http://dx.doi.org/10.1145/311535.311548

[3] ser. Chemical Industries. CRC Press, Jul 2006, ch. Non-Newtonian FluidBehavior, pp. 9–48, 0. [Online]. Available: http://dx.doi.org/10.1201/9781420015386.ch2

[4] T. Phillips and G. Roberts, "Lattice boltzmann models for non-newtonian flows," *IMA journal of applied mathematics*, vol. 76, no. 5, pp. 790–816, 2011.

[5] D. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice Boltzmann Models: An Introduction*, ser. Lattice-gas Cellular Automata and Lattice Boltzmann Models: An Introduction. Springer, 2000, no. no. 1725. [Online]. Available: http://books.google.co.in/books?id=cHcpWgxAUu8C

[6] P. L. Bhatnagar, E. P. Gross, and M. Krook, "A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems," *Phys. Rev.*, vol. 94, pp. 511–525, May 1954. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRev.94.511

[7] U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, "Lattice gas hydrodynamics in two and three dimensions," *Complex Syst.*, vol. 1, pp. 649–707, 1987.

[8] S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, J. Fagerberg, D. Mowery, and R. Nelson, Eds. Oxford: Oxford University Press, 2001.

[9] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, "A hybrid particle level set method for improved interface capturing," *Journal of Computational Physics*, vol. 183, no. 1, pp. 83 – 116, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999102971664

[10] D. Enright, F. Losasso, and R. Fedkiw, "A fast and accurate semi-lagrangian particle level set method," *Computers & structures*, vol. 83, no. 6, pp. 479–490, 2005.

[11] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen, "Physically based modeling and animation of fire," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 721–728, July 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566643

[12] F. Losasso, J. O. Talton, N. Kwatra, and R. Fedkiw, "Two-way coupled sph and particle level set fluid simulation," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 14, no. 4, pp. 797–804, 2008.

[13] M. Lentine, J. T. Grétarsson, and R. Fedkiw, "An unconditionally stable fully conservative semi-lagrangian method," *J. Comput. Phys.*, vol. 230, no. 8, pp. 2857–2879, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jcp.2010.12.036

[14] M. Desbrun and M.-P. Gascuel, "Smoothed particles: A new paradigm for animating highly deformable bodies," in *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*. New York, NY, USA: Springer-Verlag New York, Inc., 1996, pp. 61–76. [Online]. Available: http://dl.acm.org/citation.cfm?id=274976.274981

[15] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=846276.846298

[16] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 965–972, July 2005. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073298

[17] N. Chentanez, B. E. Feldman, F. Labelle, J. F. O'Brien, and J. R. Shewchuk, "Liquid simulation on lattice-based tetrahedral meshes," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2007, pp. 219–228. [Online]. Available: http://dl.acm.org/citation.cfm?id=1272690.1272720

[18] S. Chen and G. D. Doolen, "Lattice boltzmann method for fluid flows," *Annual Review of Fluid Mechanics*, vol. 30, no. 1, pp. 329–364, 1998. [Online]. Available: http://dx.doi.org/10.1146/annurev.fluid.30.1.329

[19] N. Thuerey and U. Ruede, "Optimized Free Surface Fluids on Adaptive Grids with the Lattice Boltzmann Method," *Poster, SIGGRAPH*, 2005.

[20] ——, "Free Surface Lattice-Boltzmann fluid simulations with and without level sets," *Proc. of Vision, Modelling, and Visualization VMV*, 2004.

[21] N. Thuerey, K. Iglberger, and U. Ruede, "Free Surface Flows with Moving and Deforming Objects for LBM," *Proceedings of Vision, Modeling and Visualization 2006*, pp. 193–200, Nov 2006.

[22] N. Thuerey and U. Ruede, "Stable free surface flows with the lattice boltzmann method on adaptively coarsened grids," *Computing and Visualization in Science*, vol. 12, no. 5, pp. 247–263, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00791-008-0090-4

[23] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien, "A method for animating viscoelastic fluids," *ACM Transactions on Graphics (Proc. of ACM SIGGRAPH 2004)*, vol. 23, no. 3, pp. 463–468, 2004. [Online]. Available: http://graphics.cs.berkeley.edu/papers/Goktekin-AMF-2004-08/

[24] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," in *Symposium on Computer Animation*, 2005, pp. 219–228.

[25] J. Boyd, J. Buick, and S. Green, "A second-order accurate lattice boltzmann non-newtonian flow model," *Journal of Physics A: Mathematical and General*, vol. 39, no. 46, p. 14241, 2006. [Online]. Available: http://stacks.iop.org/0305-4470/39/i=46/a=001

[26] S. Gabbanelli, G. Drazer, and J. Koplik, "Lattice boltzmann method for non-newtonian (power-law) fluids," *Phys. Rev. E*, vol. 72, p. 046312, Oct 2005. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.72.046312

[27] G. Kaehler and A. Wagner, "Derivation of Hydrodynamics for Multi-Relaxation Time Lattice Boltzmann using the Moment-Approach, Tech. Rep. arXiv:1011.1510, Nov 2010.

[28] Q. Chen, X. B. Zhang, and J. F. Zhang, "Numerical simulation of neumann boundary condition in the thermal lattice boltzmann model," *International Journal of Modern Physics C*, vol. 25, no. 08, p. 1450027, 2014. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129183114500272