Delft University of Technology
Faculty of Applied Sciences

Bachelor Thesis

Investigation of multiphase flow patterns through a T-shaped microchannel using the Lattice Boltzmann Method

Author:
Maarten Kurstjens

August 15, 2021

Supervisor: Dr. ir. M. Rohde
Department: Radiation Science and Technology (RST)
Section: Reactor Physics and Nuclear Materials (RPNM)
Group: Transport Phenomena in Nuclear Applications

Abstract

The field of microfluidics has grown rapidly over recent years. It is used in numerous fields, such as liquid-liquid solvent extraction, bio-medicine, chemical synthesis and enhanced oil recovery. The small scale on which microfluidics operates makes it safe to work with and easy to control the flow. Inside microchannels, two fluids can flow through the channel in different patterns. Flow patterns such as slug flow, droplet flow and parallel flow can be observed. These flow patterns have different interface areas and therefore different heat and mass transfer phenomena. For certain applications some flow patterns are therefore more desirable than others, thus controlling the flow pattern inside a microfluidic device is beneficial in many applications. To control the desired flow pattern, numerous parameters can be adjusted. Fluid properties such as density, viscosity and flow rate and channel properties such as shape, geometries and material all affect the flow pattern within the channel. This research investigates whether the color-gradient Rothman-Keller Lattice Boltzmann model can simulate two-phase flow through a T-shaped microchannel by analysing simulated flow patterns for different inlet flow rates and comparing the results to findings in previous experimental research.

The color-gradient Rothman-Keller model is adopted from the Lattice Boltzmann method, a numerical method in the field of Computational Fluid Dynamics (CFD). Using numerical methods instead of an experimental setup saves time needed to conduct experiments and the costs that come with fabricating different micro-channels. The Lattice Boltzmann method works by discretizing the Boltzmann equation to find a numerical solution to that equation, which can then be used to determine macroscopic fluid properties such as density and momentum through the Chapman-Enskog expansion.

The results of the numerical simulations of two-phase flow of water and toluene through a T-shaped channel with a width of 400 μm show that slug flow, droplet flow, slug-droplet flow and parallel flow patterns are observed at different inlet flow rates and flow rate ratios. Slug flow is observed for low inlet flow rates and ratios, droplet flow is observed for medium inlet flow rates and ratios and slug-droplet and parallel flow are observed for relatively high inlet flow rates and ratios.

When comparing these results to the experimental findings of Kashid et al, there are some key differences. The rates at which these flow patterns were observed in the experimental research were an order of magnitude higher than the flow rates used in this research. Kashid et al also found that a wide range of flow rates lead to deformed interface flow, a type of flow not observed during simulations. The model used in this research was also not able to simulate flow rate ratios lower than 5 and inlet flow rates higher than 0.03 m/s. The results of the simulations therefore do not match the results of experimental research. This is mostly due to the inaccurate way of incorporating surface tension and contact angle into the model.

For further research, different boundary condition schemes for contact angle and surface tension need to be imposed to improve the model's accuracy. A finer computational domain is required so that higher inlet flow rates can be simulated. More simulations need to be done to find the transition zones between certain flow patterns.

Contents

1 Introduction

1.1 Microfluidic flow

Throughout recent years, the research on microfluidic systems has grown rapidly. Microfluidics studies the behavior of fluids on the micro-scale. Using fluids on the micro-scale has numerous advantages. The large surface-to-volume ratio increases the speed of mass and heat transfer, both crucial in chemical processes. Due to their small volume, microfluidic systems are safe, easy to control and only small samples are sufficient for usage [1]. Microfluidic systems are therefore used in a variety of fields, such as chemical synthesis, bio-medicine and solvent extraction through gas-liquid and liquid-liquid extraction [2].

Flow behavior of one or multiple immiscible (non-mixing) fluids through microfluidic devices has been the subject of many previous studies. In the micro-scale, surface tension dominates gravity and inertia, allowing two immiscible fluids with different densities to flow side by side. Under laminar flow conditions, different flow patterns can be observed. Common flow patterns for liquid-liquid two-phase flow are parallel flow, slug flow and droplet flow, shown in Figure 1.

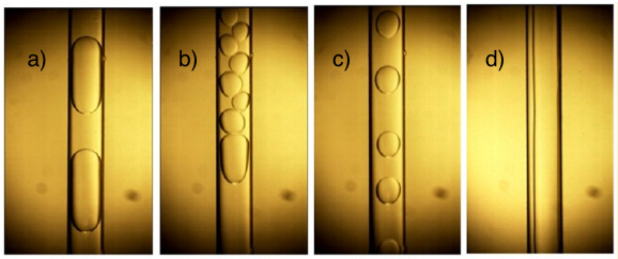
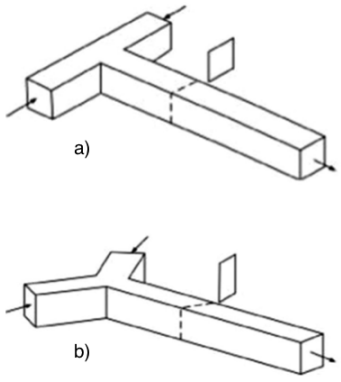
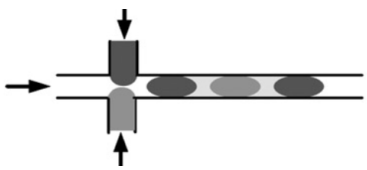


Figure 1: a) Slug flow, b) Slug and droplet flow, c) Droplet flow, d) Parallel flow [3]

Each flow has a different interface area, leading to different mass and heat transfer phenomena. For example, a high surface to volume ratio, a trait of slug flow, leads to intensified heat transfer [3]. Therefore, the performance of a microfluidic device for these transfer phenomena depends on the flow pattern inside the channel of the device [2]. To control the desired flow and therefore the performance of a microfluidic device, different parameters can be adjusted. These parameters can be divided into two groups. The first group is fluid properties, such as flow velocity, fluid density and fluid viscosity. The second group is channel properties. This group includes channel shape, dimensions and material. Three widely used channel shapes are the T-shaped channel, the Y-shaped channel and the cross-shaped channel [2]. These geometries are shown in Figure 2a and Figure 2b.



(a) a) T-shaped channel, b) Y-shaped channel [2]



(b) Cross-shaped channel [4]

The effect of these parameters on flow phenomena has been the topic of multiple studies before. Jahromi et al [5] studied the ion-pair extraction-reaction in a Y-shaped channel. They observed different flow patterns by changing the volumetric flow rates of two phases inside the Y-shaped channel, with low flow rates (1-5 μl/min) leading towards slug flow and high flow rates (40-50 μl/min) leading to parallel flow. Darekar et al [3] studied

through experiments the liquid-liquid two-phase flow through a Y-shaped channel by changing the diameter of the channel, flow rate, interfacial tension of the fluids and the hydrophobicity of the channel wall. They also found that lower flow rates of the two fluids leads to slug flow, while higher flow rates lead to parallel flow. In addition, they found that decreasing the diameter of the microchannel increases the tendency towards parallel flow, while increasing the channel diameter increases the tendency towards slug flow. Zhao et al [6] studied experimentally liquid-liquid two-phase mass transfer in a T-shaped channel by varying flow rate, inlet diameter, inlet location and channel width and height. Due to their choice of channel material, they observed for an opposed T-junction microchannel only different types of parallel flow and chaotic thin striations flow, with low Reynolds numbers leading to parallel flow with a smooth or wavy interface and high Reynolds numbers leading to chaotic thin striations flow. Kashid et al [7] studied the two-phase flow patterns in T-, Y- and concentric microchannels for different inlet flow rates. They also found that low flow rates leads to slug flow in all channel geometries, with the Y-channel and concentric channels leading towards parallel flow at slightly higher flow rates than the T-shaped channel.

1.2 Computational Fluid Dynamics

One way to analyze the effect of the outlined parameters on microfluidic flow is through numerical simulation. The field of Computational Fluid Dynamics (CFD) is used to find solutions to these simulations. Using CFD has numerous advantages over experimental research. No experimental set-up is needed, eliminating the costs of materials and time used to produce the needed equipment and conduct the experiment.

There are numerous methods to solve multiphase flows, such as the phase field (PF) method, the volume-of-fluid (VOF) method, the level-set (LS) method, the density functional method (DFM) and Lattice Boltzmann (LB) methods. The PF method describes the interface as a transition layer where unstable mixtures are stabilized. To ensure accuracy, this interface layer needs to be thin compared to other hydrodynamic length scales. The VOF method tracks the volume of each fluid in cells that belong to the interface. The VOF algorithms consist of interface reconstruction, advection and surface tension. The LS method defines the interface by a level-set function and is used often due its simplicity. LB methods model phase segregation by inter-particle interactions and therefore interface tracking is not always needed [8].

During this research, LB methods will be used to simulate multiphase flow through a micro-channel. Due to their simplicity and efficiency, LB methods are widely used to simulate multiphase flows. LB methods allow artificial compressibility to solve the incrompressible Navier-Stokes equation and they do not require the solving of the Poisson equation for pressure, a very computationally expensive process, both qualities contributing to the simplicity of LB methods. There are already multiple LB methods available for simulating multiphase flow [9]. This research will use the color-gradient Rothman-Keller model.

1.3 Thesis outline

The aim of this research is to analyze the flow patterns for liquid-liquid two phase flow of two immiscible fluids in a T-shaped microchannel in 2 dimensions using the color-gradient Rothman-Keller (RK) mutliphase model. This gives rise to the following research questions:

- Is the color-gradient RK Model able to simulate multiphase flow through a T-shaped channel?
  - What model parameters influence the accuracy, stability and type of flow pattern observed inside the channel?
  - Is the model able to simulate the same flow patterns as found in previous experimental research for certain fluid and channel parameters?

To answer these questions, a suitable multiphase LB model within the LB method will be chosen that works best with these parameters.

Chapter 2 of this thesis elaborates on the theoretical background of the research. Basic fluid dynamics and the Lattice Boltzmann methods will be discussed. Chapter 3 describes in depth the Lattice Boltzmann method that will be used during this research. Chapter 4 will explain the results of the numerical simulations, giving an overview of the different flow patterns observed under certain channel dimensions. Chapter 5 gives the conclusion of this research. Lastly, chapter 6 gives recommendations for further research.

2 Theoretical Background

In this chapter, the theory needed to conduct this research is briefly explained. Section 2.1 focuses on fluid dynamics, explaining the governing equations, frequently used dimensionless numbers and the wetting phenomenon. Section 2.2 looks at the common Lattice Boltzmann method, explaining how the method works and how different LB models extend the common LB method.

2.1 Fluid dynamics

The field of fluid dynamics looks at the macroscopic phenomena of fluid motion. The change of mass of a fluid element with fixed volume V and density ρ is given by the continuity equation [9]:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{1}$$

The momentum of this fluid element can change due to 3 factors: the flow of momentum in and out of the fluid element, differences in stresses p and external body forces $\mathbf{F}$. If we consider the density to be constant so that the flow is incompressible, the momentum equation for a fluid is given by the incompressible Navier-Stokes equation [9]:

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \eta \Delta \mathbf{u} + \mathbf{F} \tag{2}$$

Here, $\mathbf{u}$ is the fluid velocity, p is the pressure and η is the shear viscosity of the fluid. $\frac{D}{Dt}$ is called the material derivative and Δ the Laplace operator:

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla \tag{3}$$

$$\Delta = \nabla \cdot \nabla \tag{4}$$

One could also look at the motion of fluids on a microscopic level by tracking individual fluid molecules, governed by Newton's dynamics. In between the macro and micro scales lies the mesoscopic scale, the scale at which Lattice Boltmann models operate. Fluid dynamics at this scale are described by kinetic theory. In kinetic theory, 'families' of molecules are tracked. Section 2.2.1 will elaborate more on this.

2.1.1 Dimensionless numbers

In fluid dynamics, dimensionless numbers play a significant role. They indicate the interaction between forces that affect flow, such as buoyancy, gravitational, inertial, viscous and interfacial forces. On the microscale, buoyancy and gravitational forces can be neglected, since the dimensionless numbers that demonstrate the effect of these forces, the Grashof number (buoyancy to viscous forces) and the Bond number (gravitational to interfacial), scale with channel dimension to the third and second power respectively and are therefore very small [4]. Frequently used dimensionless numbers in microfluidics are the Reynolds number and capillary number.

The Reynolds number is a measure of the ratio of inertial and viscous forces. It is used to describe if the flow is turbulent (chaotic) or laminar (stream-lined), the first associated with high and the latter with small Reynolds numbers. The Reynolds number is given as:

$$Re = \frac{Inertia}{Viscous} = \frac{Lu}{\nu} \tag{5}$$

Here, L is the characteristic length of the channel, u is the flow velocity and ν is the kinematic viscosity. ν can be rewritten as $\nu = \frac{\eta}{\rho}$, where η is the dynamic viscosity and ρ the density of the fluid. With microchannels being very small, the characteristic length of the channel is very small, resulting in fairly small Reynolds numbers. This shows that viscous forces are more dominant in microfluidic flow than inertial forces. Therefore, laminar flow is expected in microchannels.

The capillary number is the ratio between viscous and interfacial forces. It is given as:

$$Ca = \frac{Viscous}{Interfacial} = \frac{\eta u}{\sigma} \tag{6}$$

Here, σ is the interfacial tension. The capillary number is completely controlled by fluid properties and flow rate, not by channel dimensions like the Reynolds number. It is used to describe fluid phenomena like fluid formation and therefore fluid flow patterns [4].

2.1.2 Wetting and contact angle

Wetting describes the behavior of fluids on the interface between a fluid and a solid. Fluids stick to solids differently through adhesive forces between fluid particles and molecules in the solid surface and cohesive forces between the fluid molecules themselves. These forces differ for different fluids, a property specified in the contact angle. Some fluids spread out over the solid, others do not stick to the solid at all. Contact angle θ of fluid 1 that is sticking to a wall and surrounded by fluid 2 is given as [10]:

$$cos(\theta) = \frac{\sigma_{s1} - \sigma_{s2}}{\sigma_{12}} \tag{7}$$

Here, σ_{s1} is the surface tension between fluid 1 and the solid wall, σ_{s2} is the surface tension between fluid 2 and the solid wall and σ_{12} is the surface tension between fluid 1 and fluid 2. Contact angle is visualised in Figure 3.

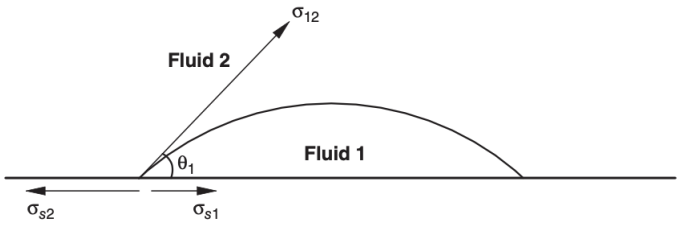


Figure 3: Contact angle [10]

How fluids interact with the solid wall of a channel affects the type of flow inside the channel. When $\theta < 90°$, a fluid is called hydrophilic, meaning it wets the solid wall well. When $\theta > 90°$, a fluid is called hydrophobic, meaning a droplet tends to detach from the solid surface.

2.2 Lattice Boltzmann Method

2.2.1 Introduction

As stated in paragraph 2.1, Lattice Boltzmann models operate on the mesoscopic scale Fluid dynamics at this scale are described by kinetic theory. In kinetic theory, 'families' of molecules are tracked. These families are described by the particle distribution function $f(\mathbf{x}, \boldsymbol{\xi}, t)$. Here, $\boldsymbol{\xi}$ is the microscopic particle velocity. Macroscopic fluid variables like density ρ and momentum density $\rho\mathbf{u}$ can be retrieved from the particle density function by integrating over f:

$$\rho(\mathbf{x}, t) = \int f(\mathbf{x}, \boldsymbol{\xi}, t) d^3\xi \tag{8}$$

$$\rho(\mathbf{x}, t)\mathbf{u} = \int \boldsymbol{\xi} f(\mathbf{x}, \boldsymbol{\xi}, t) d^3\xi \tag{9}$$

If a fluid is left alone long enough, the particle distribution function will reach an equilibrium distribution $f^{eq}(\mathbf{x}, \boldsymbol{\xi}, t)$. To see how f evolves over time, the Boltzmann equation is used:

$$\frac{\partial f}{\partial t} + \boldsymbol{\xi} \cdot \nabla f + \mathbf{F} \cdot \nabla_{\xi} f = \Omega(f) \tag{10}$$

$\Omega(f)$ is called the collision operator. This operator describes the local redistribution of f due to collisions with other particles. The collision operator used in most LB methods is called the BGK collision operator, named after its inventors Bhatnagar, Gross and Krook. The BGK collision operator is given by:

$$\Omega(f) = -\frac{1}{\tau}(f - f^{eq}) \tag{11}$$

τ is called the relaxation time, which is related to the kinematic viscosity of the fluid. τ determines the speed at which the distribution function goes to equilibrium.

Through the use of the Boltzmann equation, macroscopic variables can be retrieved from the particle distribution function to solve the Navier-Stokes equation through the Chapman-Enskog expansion [9].

Lattice Boltzmann methods come forth from the use of cellular automata [10]. Cellular automata works through the discretization of space into individual cells, with each cell having a particular state. These states are updated each time step through a set of conditions that take as input the states of neighboring cells. Lattice Botlzmann methods expand on this model.

The first step for using the Lattice Boltzmann method to solve the Boltzmann equation is discretising velocity space, physical space and time to obtain the discrete distribution function $f_i(\mathbf{x}, t)$ [9]. The discrete distribution function describes a probability density of particles with a certain velocity $\mathbf{e}_i$ at position $\mathbf{x}$ and time t. These velocities come from a discrete set of velocities, or velocity set, determined by the dimension used in the model. The i-subscript in $f_i(\mathbf{x}, t)$ and $\mathbf{e}_i$ refer to the ith velocity in the velocity set. The discrete distribution functions are defined at lattice points $\mathbf{x}$ in space, positioned a distance Δx, or one lattice unit lu, apart. f_i is also only defined at discretized times t, separated by a time Δt, or one time step ts.

Retrieving macroscopic fluid variables is now done by summing the discrete distribution functions over i:

$$\rho(\mathbf{x}, t) = \sum_i f_i(\mathbf{x}, t) \tag{12}$$

$$\rho\mathbf{u}(\mathbf{x}, t) = \sum_i \mathbf{e}_i f_i(\mathbf{x}, t) \tag{13}$$

Discretising the Boltzmann equation gives the Lattice Boltzmann equation [10]:

$$f_i(\mathbf{x} + \mathbf{e}_i\Delta t, t + \Delta t) = f_i(\mathbf{x}, t) - \frac{1}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) + S_i(\mathbf{x}, t) \tag{14}$$

Here, S_i is an added source term which differs depending on the forcing term. The relaxation time τ is related to the kinematic viscosity by:

$$\nu = c_s^2(\tau - 0.5)\Delta t \tag{15}$$

c_s is called the sound speed, which also depends on which velocity set is used. f_i^{eq} is calculated as follows:

$$f_i^{eq}(\mathbf{x}, t) = w_i\rho\left[1 + \frac{\mathbf{e}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{(\mathbf{u})^2}{2c_s^2}\right] \tag{16}$$

Here, w_i are weights, also given by the chosen velocity sets.

These velocity sets are specified using DnQm notation, where n stands for the desired space dimension of the model and m stands for the number of velocities in the velocity set. As stated before, each velocity set has a different set of weights w_i, velocities $\mathbf{e}_i$ and sound speed c_s^2. The number of velocities in the set corresponds to the number of neighboring lattice sites that each particle distribution at a certain lattice point has. This is shown in Figure 4. Here, the most used velocity models D2Q7, D2Q9, D3Q15 and D3Q19, are given.

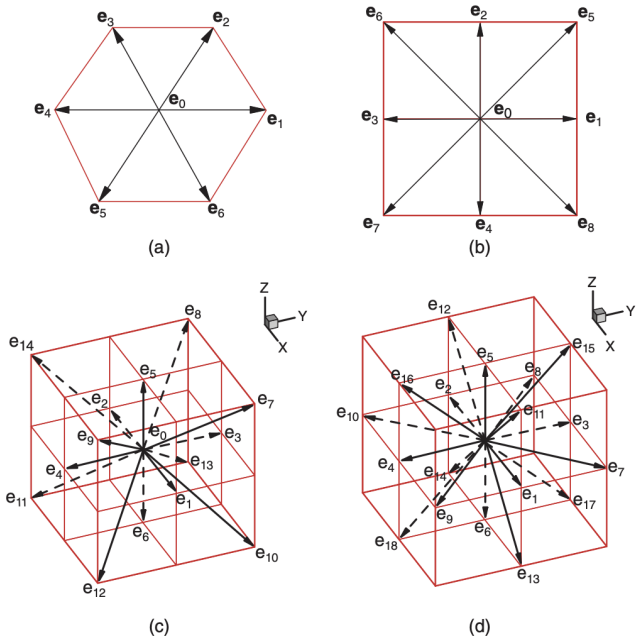


Figure 4: Discrete velocity models (a) D2Q7, (b) D2Q9, (c) D3Q15, (d) D3Q19 [10]

The weights w_i and sound speed c_s^2 of each model are given in Table 1.

Model	w_i	c_s^2
D2Q7	$\frac{1}{2}$ $(i = 0)$, $\frac{1}{12}$ $(i = 1, ..., 6)$	$\frac{c^2}{4}$
D2Q9	$\frac{4}{9}$ $(i = 0)$, $\frac{1}{9}$ $(i = 1, 2, 3, 4)$, $\frac{1}{36}$ $(i = 5, 6, 7, 8)$	$\frac{c^2}{3}$
D3Q15	$\frac{2}{9}$ $(i = 0)$, $\frac{1}{9}$ $(i = 1, ..., 6)$, $\frac{1}{72}$ $(i = 7, ..., 14)$	$\frac{c^2}{3}$
D3Q19	$\frac{1}{3}$ $(i = 0)$, $\frac{1}{18}$ $(i = 1, ..., 6)$, $\frac{1}{36}$ $(i = 7, ..., 18)$	$\frac{c^2}{3}$

Table 1: Weights and sound speeds of different velocity models [10]

Here, c is called the lattice speed given as $c = \frac{\Delta x}{\Delta t}$. Often, Δx and Δt are both 1, giving a lattice speed of $c = 1$. In this research, the D2Q9 model will be used. The velocity set for this model is given as follows:

$$[\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_6, \mathbf{e}_7, \mathbf{e}_8] = c \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}$$

2.2.2 Streaming and Colliding

To briefly summarize, the Lattice Boltzmann method finds a numerical solution to the Boltzmann equation by simulating 'families' of particles, discrete distribution functions, at different lattice points in space, separated by 1 lu, at different points in time, separated by 1 ts. The flow of the fluid is simulated through the stream and collide steps between nodes[9].

Lets assume $S_i = 0$. The Lattice Boltzmann equation then becomes:

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta t, t + \Delta t) = f_i(\mathbf{x}, t) - \frac{1}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) \tag{17}$$

This equation can be divided into 2 separate parts. The first is the collision step, given by:

$$f_i^*(\mathbf{x}, t) = f_i(\mathbf{x}, t) - \frac{1}{\tau}(f_i(\mathbf{x}, t) - f_i^{eq}(\mathbf{x}, t)) \tag{18}$$

Here, f_i^* is the distribution function after the collision step. Prior to the collision step, the macroscopic variables ρ and $\mathbf{u}$ are calculated and used to calculate f_i^{eq} using equation 16. The collision step considers all possible outcomes of a collision between two particles. It takes into account the relaxation time τ. As stated

in section 2.1, τ determines the speed at which the distribution function goes to equilibrium. If τ is big, it will take longer to reach equilibrium, which means it is resisting flow. This can be shown through relation 15, which states that a larger τ corresponds to a larger viscosity, a fluid property which is a measure of how a fluid resists deformation. The collision step therefore takes in account the resistance of a fluid to flow through its viscosity. To improve the stability of the model, τ should be bigger and not be too close to 0.5 [9].

After the collision step, the post collision distribution function f_i^* is streamed to the neighboring lattice nodes. This is the second part, known as the streaming step, which is given by:

$$f_i(\mathbf{x} + \mathbf{e}_i \Delta t, t + \Delta t) = f_i^*(\mathbf{x}, t) \tag{19}$$

The streaming step is visualised by Figure 5.

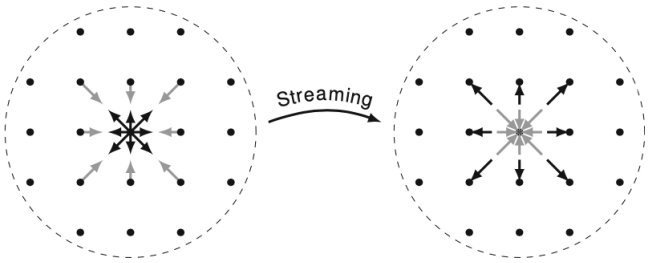


Figure 5: Particles streaming from one node to neighboring nodes [9]

These steps are then repeated in the next time step. To ensure the model suffices for physical boundaries such as channel walls and geometries, boundary conditions are imposed to the system. Paragraph 3.2 explains the use of boundary conditions in this research.

These steps together create a loop, visualised in Figure 6.

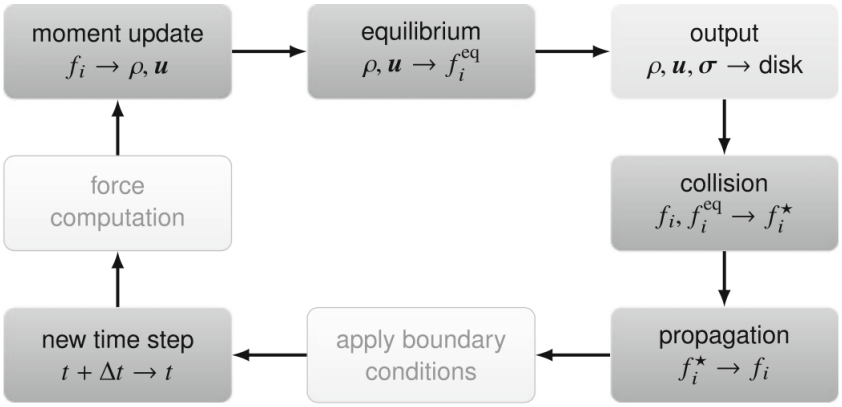


Figure 6: Lattice Botlzmann Algorithm [9]

This is the framework for the common Lattice Boltzmann method. Different models within the Lattice Boltzmann method build upon this framework to ensure that they are as accurate as possible to simulate the type of flow requested.

2.2.3 Unit conversion

For a Lattice Boltzmann model to work, physical units have to be converted into dimensionless units, such as lattice units and time steps. This is called the non-dimensionalisation of physical parameters [9]. This is done with the use of chosen conversion factors and the law of similarity. All physical parameters, such as length, time, density, viscosity, need to be converted into dimensionless Lattice Boltzmann (LB) units using conversion factors. For example, a physical length is converted into a non-dimensional length as follows:

$$l^* = \frac{l}{C_l} \tag{20}$$

Here, l is the physical length, l^* is the LB length and C_l is the length conversion factor. The law of similarity is useful to calculate LB parameters from other LB parameters. It states that dimensionless numbers calculated using physical parameters are equal to the same dimensionless number calculated using LB parameters [9]. The Reynolds number is used as an example:

$$Re = \frac{Lu}{\nu} = \frac{L^* u^*}{\nu^*} \tag{21}$$

In short, if one wishes to know for example the density ρ after an amount of time steps, equation 12 is used to calculate the LB density. Next the conversion factor C_ρ is used to calculate the physical density.

When LB models are discussed in this paper, all parameters in the equations are LB parameters, making the * obsolete. It will be specified otherwise.

2.2.4 Different Multiphase LB Models

Multiple multiphase LB Models have been developed and adjusted over recent years to simulate multiphase flow. Popular models are the color-gradient model, which is based on the Rothman-Keller (RK) multiphase lattice gas model, the multiphase Shan-Chen (SC) model and the free-energy (FE) model [10]. These 3 models are briefly introduced.

Color-gradient RK model: In the color-gradient RK model, first proposed by Rothman and Keller in 1988 [11], one fluid is given a blue color and the other a red color. Each fluid is represented by its own distribution function. Next to the common collision term in the LB model, there is a second collision term added, together with a re-coloring step. With the color-gradient model, surface tension and the viscosity ratio between the blue and red fluid can be adjusted independently. However, it is not able to handle high-density ratio simulations [10].

Multiphase Shan-Chen model: In the Multiphase Shan-Chen model, proposed by X. Shan and H. Chen in 1993 [12], a repulsive force acts as the interaction between particles, which leads to phase separation and interface maintenance. It incorporates an attractive forcing term that introduces a non-ideal non-monotonic equation of state instead of the ideal gas equation of state. Just like in the color-gradient model, each fluid component is represented by its own distribution function. It handles high-density ratios well, but the density and viscosity ratios and surface tension cannot be adjusted independently [10].

Free-energy model: The free-energy model was proposed by Swift et al in 1996 [13]. Where the RK model and the SC model are 'bottom-up' approaches, where microscopic interactions between fluid elements are specified, the FE model uses a 'top-down' approach. It starts with a macroscopic concept, the free energy of the fluids. From the free energies, the phase separation and surface tension can be derived [9]. The model is often used for increased thermodynamic consistency. It is able to simulate high-density ratio cases, but it has to solve the Poisson-equation, which decreases the simplicity of the model.

Huang et al [10] made a comparison of these models. While the multiphase SC model is very efficient, it is less accurate. The RK and FE model are less efficient, but more accurate. In the RK and SC model, the contact angle between a fluid and a solid is obtained by specifying a 'wall density' for the wetting condition. The FE model uses a density gradient to implement contact angle, which is less convenient.

In this research, the color-gradient RK model will be used. It allows surface tension and viscosity ratios to be adjusted independently, which makes it easier to replicate set-ups used in experimental research so that the results of the numerical simulations of this research can be compared to the results of previous experimental research. Being able to specify a wall density makes it easier to implement a wetting condition to incorporate contact angle into the model.

3 Color-Gradient Rothman-Keller Model

In this chapter, a detailed description of of the color-gradient Rothman-Keller model is given. Section 3.1 explains how to implement the model to use during simulations. Section 3.2 explains which boundary conditions are used to simulate two-phase flow through the T-shaped microchannel. Section 3.3 explains the Python code used to run the simulations in this research. Section 3.4 will discuss the initial model parameters and fluids used in this research.

3.1 Model outline

In the RK model, the two fluids are given a color, blue and red, both represented by their own particle distribution function f_i^1 and f_i^2 [10]. The total particle distribution function is given as:

$$f_i(\mathbf{x}, t) = \sum_k f_i^k(\mathbf{x}, t), \qquad k = 1, 2 \tag{22}$$

The RK model adds an extra step to the streaming and collision step of the common LB method, the recoloring step. It also includes a second collision term, called the perturbation collision operator. This operator implements surface tension into the model. The recoloring step encourages color separation at the interface while still conserving mass and momentum [14]. It introduces a color gradient parameter $\mathbf{f}$ that works normal to the interface, adding populations normal to the interface and removing them parallel to the interface. The RK model algorithm then works as follows.

First, the streaming step is given as:

$$f_i^k(\mathbf{x} + \mathbf{e}_i\Delta t, t + \Delta t) = f_i^{k+}(\mathbf{x}, t) \tag{23}$$

Here, f_i^{k+} is the particle distribution function after the recoloring step. The velocity set from the D2Q9 model is used for $\mathbf{e}_i$. After the streaming step, the collision step is applied. As mentioned before, the RK model adds a second collision term next to the LB BGK collision operator called the perturbation operator. The RK collision step is given as:

$$f_i^{k*}(\mathbf{x}, t) = f_i^k(\mathbf{x}, t) + (\Omega_i^k)^1 + (\Omega_i^k)^2 \tag{24}$$

Here, $(\Omega_i^k)^1$ and $(\Omega_i^k)^2$ are the two collision terms. The first term is the BGK operator for multiple fluids, given as:

$$(\Omega_i^k)^1 = -\frac{1}{\tau}(f_i^k(\mathbf{x}, t) - f_i^{k,eq}(\mathbf{x}, t)) \tag{25}$$

The equilibrium distribution function $f_i^{k,eq}(\mathbf{x}, t)$ of fluid k is calculated as:

$$f_i^{k,eq}(\mathbf{x}, t) = \rho_k\Big(C_i + w_i\left[\frac{\mathbf{e}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{(\mathbf{u})^2}{2c_s^2}\right]\Big) \tag{26}$$

Here, w_i is again given by the D2Q9 velocity model. ρ_k is the density of fluid k, calculated by summing over f_i^k:

$$\rho_k = \sum_i f_i^k \tag{27}$$

The total density and momentum are obtained using equations 28 and 29 :

$$\rho = \sum_k \rho_k \tag{28}$$

$$\rho\mathbf{u} = \sum_k \sum_i f_i^k \mathbf{e}_i \tag{29}$$

The C_i term in equation 26 is given by:

$$C_i = \begin{cases} \alpha_k, & i = 0, \\ \frac{1-\alpha_k}{5}, & i = 1, 2, 3, 4, \\ \frac{1-\alpha_k}{20}, & i = 5, 6, 7, 8 \end{cases} \tag{30}$$

Here, α_k is the parameter used to adjust the density of the fluids. The density ratio of the two fluids is given as:

$$\frac{\rho_1}{\rho_2} = \frac{1-\alpha_2}{1-\alpha_1} \tag{31}$$

The two fluids have different relaxation times τ_1 and τ_2. At the interface, τ becomes a function of position $\mathbf{x}$. $\tau(\mathbf{x})$ is calculated using equations 32 and 33. First, the parameter $\psi(\mathbf{x})$ is defined:

$$\psi(\mathbf{x}) = \frac{\rho_1(\mathbf{x}) - \rho_2(\mathbf{x})}{\rho_1(\mathbf{x}) + \rho_2(\mathbf{x})} \tag{32}$$

$\psi(\mathbf{x})$ is used to construct a interpolation scheme for $\tau(\mathbf{x})$ so that the relaxation time transitions smoothly over the interface between the fluids:

$$\tau(\mathbf{x}) = \begin{cases} \tau_1, & \psi > \delta \\ g_1(\psi), & \delta \geq \psi > 0 \\ g_2(\psi), & 0 \geq \psi \geq -\delta \\ \tau_2, & \psi < -\delta \end{cases} \tag{33}$$

$g_1(\psi)$ and $g_2(\psi)$ are given by:

$$\begin{cases} g_1(\psi) = s_1 + s_2\psi + s_3\psi^2, \\ g_2(\psi) = t_1 + t_2\psi + t_3\psi^2, \end{cases} \tag{34}$$

with

$$\begin{cases} s_1 = t_1 = 2\frac{\tau_1\tau_2}{\tau_1+\tau_2}, \\ s_2 = 2\frac{\tau_1 - s_1}{\delta}, \\ s_3 = -\frac{s_2}{2\delta}, \\ t_2 = 2\frac{t_1 - \tau_2}{\delta}, \\ t_3 = \frac{t_2}{2\delta}, \end{cases} \tag{35}$$

The parameter δ affects the interface thickness and is normally set to $\delta = 0.98$. The kinematic viscosities of both fluids can now be calculated:

$$\nu_k = c_s^2(\tau_k - 0.5)\Delta t \tag{36}$$

These are all the terms needed to calculate the first collision term $(\Omega_i^k)^1$. To calculate the perturbation collision term $(\Omega_i^k)^2$, equation 37 is used:

$$(\Omega_i^k)^2 = \frac{A_k}{2}|\mathbf{f}|\left[w_i\frac{(\mathbf{e}_i\cdot\mathbf{f})^2}{|\mathbf{f}|^2} - B_i\right] \tag{37}$$

Here, A_k is a parameter that affects interfacial tension and B_i is a correction term given as:

$$B_i = \begin{cases} -\frac{4}{27}, & i = 0, \\ \frac{2}{27}, & i = 1,2,3,4, \\ \frac{5}{108}, & i = 5,6,7,8 \end{cases} \tag{38}$$

The parameter $\mathbf{f}$ is the color-gradient, which depends on $\mathbf{x}$ and t:

$$\mathbf{f}(\mathbf{x},t) = \sum_i \mathbf{e}_i \sum_j \left[f_j^1(\mathbf{x}+\mathbf{e}_i\Delta t, t) - f_j^2(\mathbf{x}+\mathbf{e}_i\Delta t, t)\right] \tag{39}$$

With the collision step completed, the recoloring step is implemented. The recoloring step separates the two fluids. The recoloring step is given for both fluids as:

$$f_i^{1,+} = \frac{\rho_1}{\rho}f_i^* + \beta\frac{\rho_1\rho_2}{\rho^2}f_i^{eq}(\rho, \mathbf{u}=0)cos(\lambda_i) \tag{40}$$

$$f_i^{2,+} = \frac{\rho_2}{\rho}f_i^* - \beta\frac{\rho_1\rho_2}{\rho^2}f_i^{eq}(\rho, \mathbf{u}=0)cos(\lambda_i) \tag{41}$$

Here, $f_i^{1,+}$ and $f_i^{2,+}$ are the particle distribution functions after recoloring and $f_i^* = \sum_k f_i^{k*}$ is the post-collision particle distribution function. λ_i is the angle between the color gradient $\mathbf{f}(\mathbf{x}, t)$ and the direction $\mathbf{e}_i$, with $cos(\lambda_i) = \frac{\mathbf{e}_i \cdot \mathbf{f}}{|\mathbf{e}_i||\mathbf{f}|}$. β is a parameter used to control the interface thickness.

The recoloring step completes the loop of the RK model algorithm. As a next step, $f_i^{1,+}$ and $f_i^{2,+}$ are streamed again using the streaming step, starting the loop over again. This is how multiphase flow is simulated using the color-gradient Rothman-Keller model.

Two important parameters in the RK model are $A = \sum_k \frac{A_k}{2}$ and β, the former used to determine the interfacial tension and the latter the interfacial thickness. β is a number between 0 and 1. If β is small, the interface becomes thick. This causes a diffusive interface, which has a negative impact on the accuracy of the model. However, a small β is beneficial for the stability of the model [10]. Therefore, β is usually kept between 0.5 and 0.7.

The parameter A scales with surface tension σ. Reis et al [14] define A as:

$$\sigma \approx \frac{2A}{w} \tag{42}$$

Here, w is determined using the kinematic viscosity of the mixture ν:

$$\nu = \frac{1}{3}(\frac{1}{w} - \frac{1}{2}) \tag{43}$$

3.2 Boundary conditions

3.2.1 Wall boundary condition

The first boundary condition that is imposed is the no-slip boundary condition that dictates what happens when a fluid particle hits a wall like the side of a channel. The Lattice Boltzmann method works by dividing space up into nodes spaced 1 lu apart, with on each node a particle distribution. Nodes can be divided into two groups, fluid nodes and solid nodes. Solid nodes represent obstacles like walls. Here, the particle distribution function is zero, since there are no fluid particles there. When a fluid particle streams from a fluid node to a solid node, it is bounced back to the fluid node it came from. This boundary condition is therefore called the bounce-back method. There are two different bounce-back methods, fullway bounce-back and halfway bounce-back [9]. Both methods are illustrated in Figure 7.

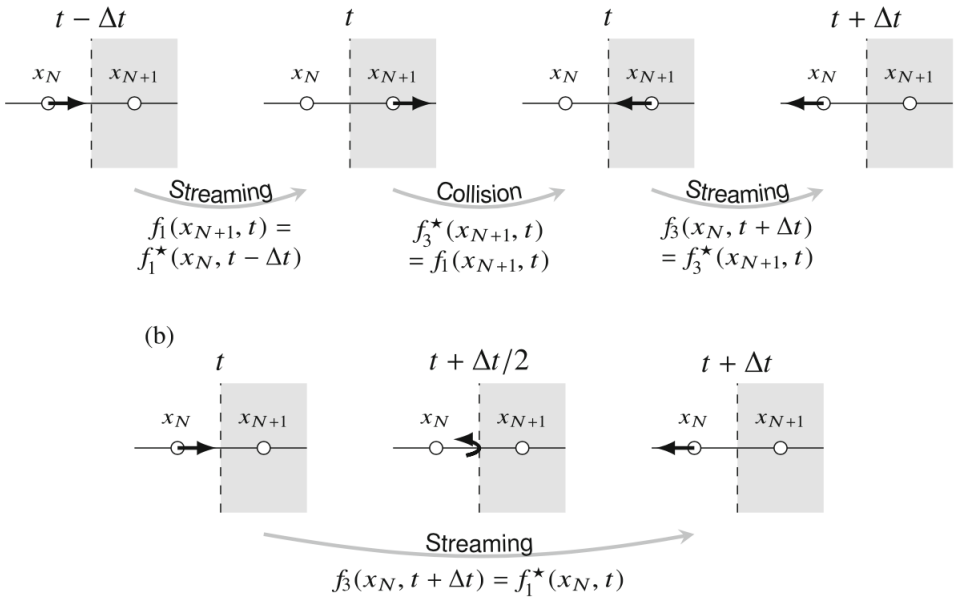


Figure 7: a) Fullway bounce-back, b) Halfway bounce-back [9]

In this figure, x_N represents a fluid node and x_{N+1} represents a solid node. The dashed line represents the

wall.

The fullway bounce-back method streams a particle the entire way to the solid node in the first time step. The particle is then collided back during the collision step to be returned to the original fluid node in the next streaming step. The method therefore takes two time steps.

When the halfway bounce-back method is used, the particle streams only halfway the distance to the solid node and then returns back to the original fluid node at $t = \frac{\Delta t}{2}$. This method only uses one streaming step and one time step, making it less time intensive than the fullway bounce-back method. Therefore, the halfway bounce-back method is used in this research. The streaming step at solid nodes is given by:

$$f_i(\mathbf{x}_b, t + \Delta t) = f_i^*(\mathbf{x}_b, t) \tag{44}$$

Here, $\mathbf{x}_b$ is the position of the fluid node where the particle is streamed from.

3.2.2 Inlet boundary condition

At the inlets of the channel, fluid has to flow into the channel with a velocity $\mathbf{u}$. This is done by simulating the inlet as a moving wall that 'pushes back' on the particle that streams to the solid node on the moving wall, giving it extra momentum. This is called the moving walls method, proposed by Ladd [15]. The moving wall formula is given by:

$$f_i(\mathbf{x}_b, t + \Delta t) = f_i^*(\mathbf{x}_b, t) - w_i \rho_w \frac{\mathbf{e}_i \cdot \mathbf{u}_w}{c_s^2} \tag{45}$$

Here, ρ_w is the wall density, often set equal to the density at $f_i^*(\mathbf{x}_b, t)$, and $\mathbf{u}_w$ is the wall velocity. The wall velocity is set equal to the inlet flow velocity. If $\mathbf{u}_w = 0$, equation 44 is obtained, which is the bounce-back equation for a resting wall.

3.2.3 Outlet boundary condition

At the outlet of the channel, the fluid flows out of the computational domain. For the nodes at the outlet that stream fluid particles out of the channel, there is an outflow boundary condition (OBC) imposed. Q. Lou et al [16] investigated three single-phase OBC's and applied them to multiphase flows to see how well each worked. An OBC is tested through two criteria: the fluid should propagate smoothly out of the outlet without affecting the interior flow and the LB model should stay accurate and stable with the OBC imposed. The investigated OBC's are the Neumann boundary condition, the convective boundary condition and the extrapolation boundary condition. They found that the convective boundary condition gave the most stable and accurate solutions.

The convective boundary condition is imposed as follows. Consider the situation illustrated in Figure 8.

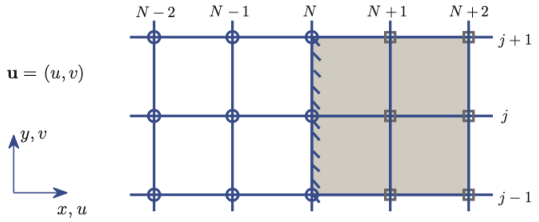


Figure 8: Nodes at outlet of a channel [16]

Here, the nodes at $x = N$ are the nodes at the outlet of the channel and nodes $N + 1$ and $N + 2$ are outside of the computational domain. The particle distribution function at node N is calculated using equation 46:

$$f_i(N, j, t + \delta t) = \frac{f_i(N, j, t) + \lambda f_i(N - 1, j, t + \delta t)}{1 + \lambda} \tag{46}$$

Here, $\lambda = \frac{U(t+\delta t)\delta t}{\delta x}$. U is the velocity normal to the boundary. There are three ways to define U:

$$U = \begin{cases} U_{max}(t) = max[u(N-1,j,t] \\ U_{ave}(t) = \frac{1}{M+1}\sum_j u(N-1,j,t) \\ U_{local}(N-1,j,t) = u(N-1,t,t) \end{cases} \tag{47}$$

$M+1$ are the number of nodes in the $(N-1)$th layer in the y-direction.

3.2.4 Wetting boundary condition

To include the wettability of the fluids in the simulation, a wetting boundary condition needs to be imposed for nodes at the boundary. On these nodes, the contact angle of the fluids has to be defined. In this research, a contact angle scheme by Huang et al [10] is used. The contact angle at a solid node is defined as:

$$\theta = arccos(\frac{\rho_{w1} - \rho_{w2}}{\rho_i}) \tag{48}$$

Here, ρ_{w1} is the density of fluid 1 at the solid node, ρ_{w2} is the density of fluid 2 at the solid node and ρ_i is the density of the majority component, which is the density at the fluid nodes surrounding the solid node.

3.3 Python Code algorithm

In this research, Python is used to implement the color-gradient RK model algorithm. The code used to run the simulations in this research can be found in section 7.2. The code works as illustrated by the flow chart in Figure 9:

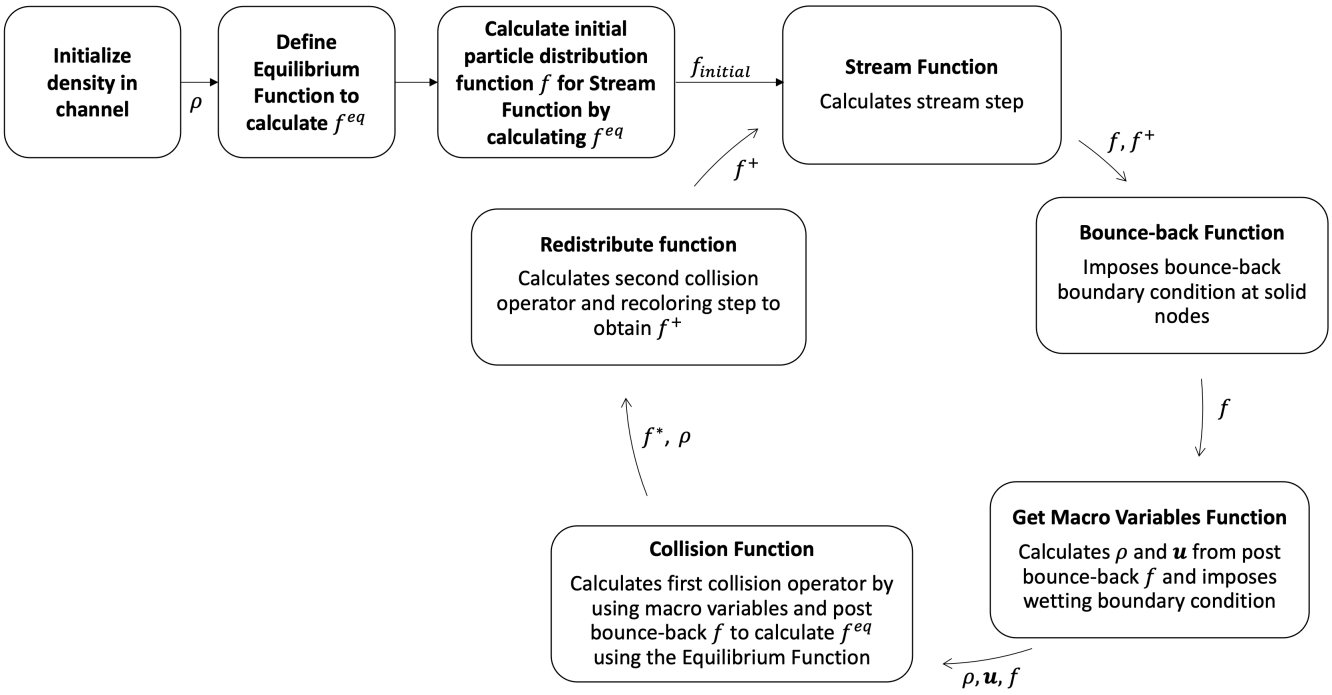


Figure 9: Flow chart of Python code structure used in simulations

3.4 Model parameters

To investigate whether the results of the model simulations match experimental results, the same channel parameters and fluids will be used as by Kashid et al [7].

The channel width of the T-shaped microchannel is therefore chosen to be $w = 400\mu m$. To recreate this T-shaped channel in the lattice domain, a channel width $w^* = 10$ lu is chosen, so that $\Delta x = 40\mu m$. The inlet length $L_{inlet} = 20$ lu and the channel length $L_{channel} = 100$ lu.

For simplicity purposes, the contact angle boundary condition is taken to be fully wetting for the blue fluid and fully non-wetting for the red fluid for all simulations. The variable A that incorporates surface tension in the model is kept constant at $A = 0.0001$ for stability and simplicity. β is kept at a value of 0.7.

The fluids used in this research are water ($\rho = 998\ \frac{kg}{m^3}$ and $\eta = 10^{-3}\ Pa \cdot s$) for the blue fluid and toluene ($\rho = 870\ \frac{kg}{m^3}$ and $\eta = 5.9 \cdot 10^{-4}\ Pa \cdot s$) for the red fluid. This gives a density ratio of 1.15 and a viscosity ratio of 1.69. The surface tension $\sigma = 0.036\ N/m$.

4 Results

4.1 Poiseuille flow

To check whether the RK model algorithm works, an example for mutliphase Poiseuille flow is worked out. Huang et al [10] gave a detailed description of this example using FORTRAN. In this example, illustrated in Figure 10, the velocity profile of annular two-phase Poiseuille flow is worked out.

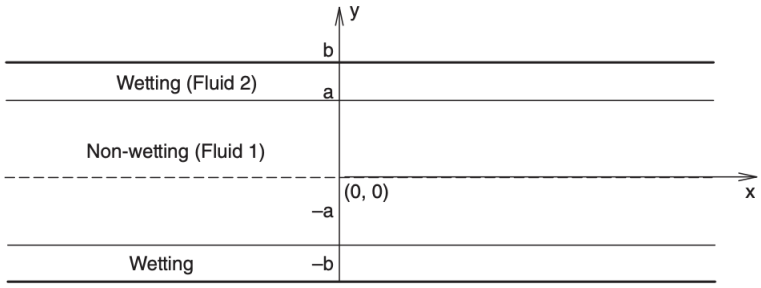


Figure 10: Two-phase Poisseuille flow of two immiscible fluids in a 2D channel. The two interfaces are positioned at $y = a$ and $y = -a$ and the channel walls are positioned at $y = b$ and $y = -b$ [10]

The velocity profile can also be solved analytically, so the model can be validated by comparing the numerical and analytical solution. The example is worked out with the density and viscosity ratio both equal to 1.5, $\beta = 0.5$ and $A = 0.0001$. The channel width is 101 lu and the channel length is 4 lu. The results of this example are shown in Figure 11.

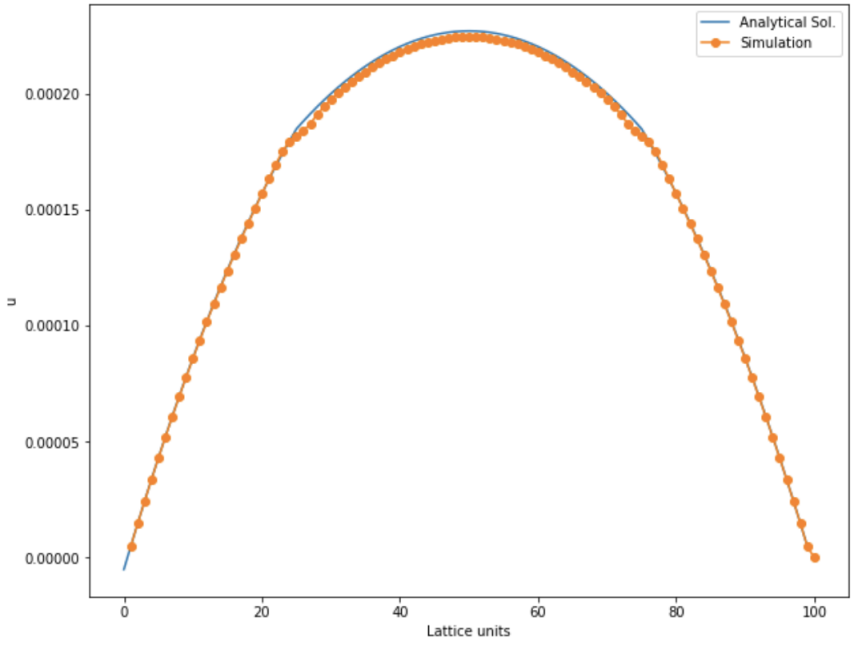


Figure 11: Plot of analytical and numerical solution for the velocity profile of two-phase Poiseuille flow

From this plot, it can be seen that the RK model used in this research can correctly simulate two-phase Poiseuille flow. The biggest inaccuracy happens at the two interfaces between the two fluids at 25 lu and 75 lu (positions $y = a$ and $y = -a$ in Figure 10). The two 'dips' in the numerical solution are due to the pressure difference that is created by the difference in density of the two fluids. Since the Lattice Boltzmann method does not track the interface, which benefits computational efficiency, the interface between the two fluids in the numerical solution is diffusive, while the interface in the analytical solution is sharp.

4.2 Flow patterns inside T-shaped microchannel

Figure 12 shows the T-shaped channel at $t = 0$. The first inlet and the channel are filled with the blue fluid, while the second inlet is filled with the red fluid. In experimental set-ups, the channel is first flushed with water before the organic fluid is pumped through. This initial setup therefore best resembles the experimental set-up.

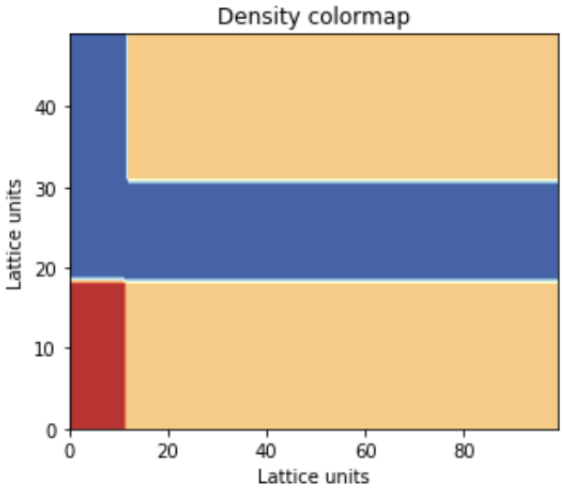


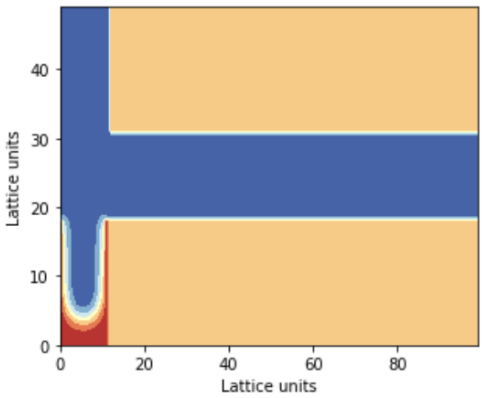
Figure 12: T-shaped channel set-up at $t = 0$

To simulate different flow patterns inside the channel, the inlet flow rates u_w and u_t of water and toluene respectively will be varied. To match the density and viscosity ratios of water and toluene in the lattice domain, α and τ for the fluids are chosen to be $\alpha_w = 0.33$, $\alpha_t = 0.42$, $\tau_w = 0.75$ and $\tau_t = 0.67$. The used inlet flow rates in lattice units are given in Table 2. The conversion factor for u is $C_u = 0.3\ m/s$. Conversion of all parameters can be found in section 7.1:

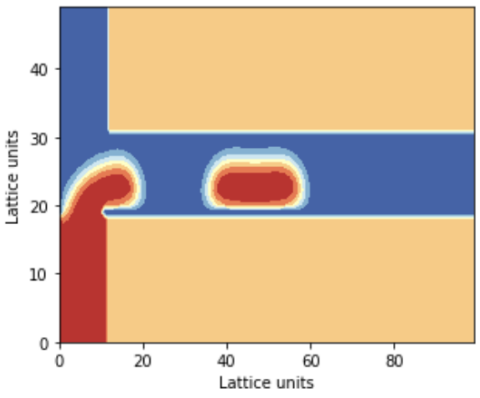
Case	u_w	u_t	$\frac{u_t}{u_w}$	Re_w	Re_t
a	0.0032	0.0032	1	0.38	0.56
b	0.00032	0.0016	5	0.04	0.28
c	0.0032	0.016	5	0.38	2.82
d	0.00032	0.0032	10	0.04	0.56
e	0.0032	0.032	10	0.38	5.65
f	0.00032	0.0064	20	0.04	1.13
g	0.0032	0.064	20	0.38	11.29
h	0.008	0.16	20	0.96	28.2

Table 2: Fluid flow rates in lattice units, flow ratios and Reynolds numbers used in simulations

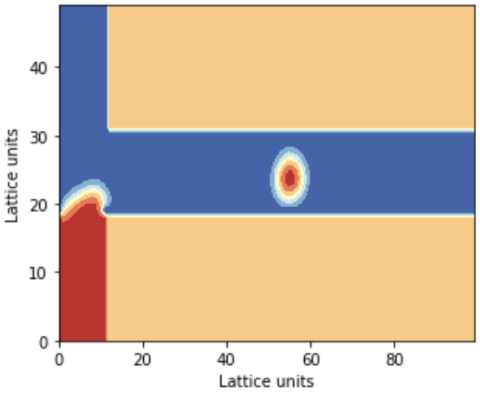
The results of each case are shown in the figures below.



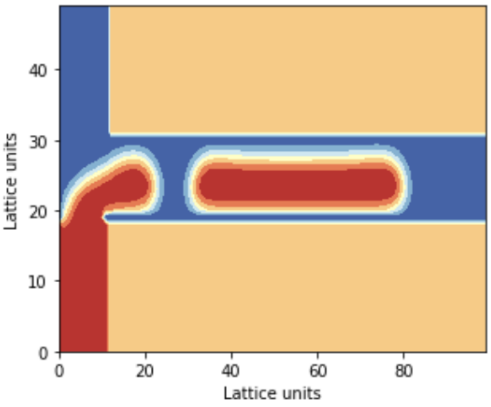
(a) Case a) after $t = 1.86\ s$, water intrudes into inlet of toluene.



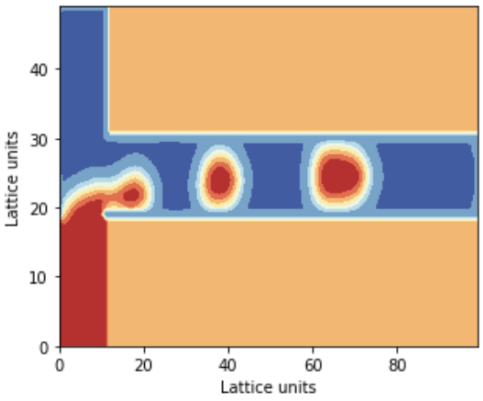
(b) Case b) after $t = 19.68\ s$, slug flow.



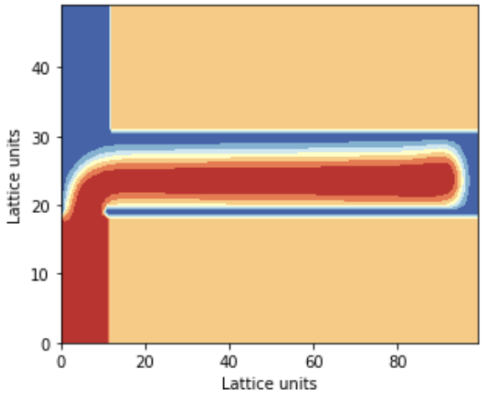
(c) Case c) after $t = 3.33\ s$, droplet flow.



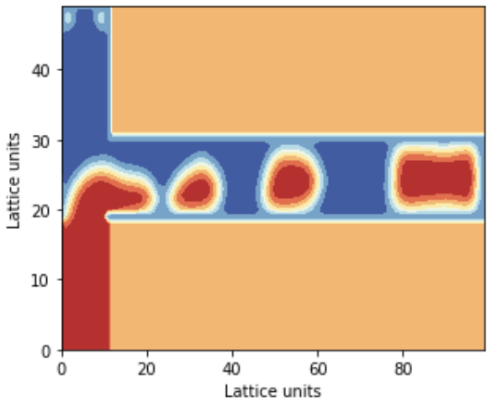
(d) Case d) after $t = 12.64\ s$, slug flow.



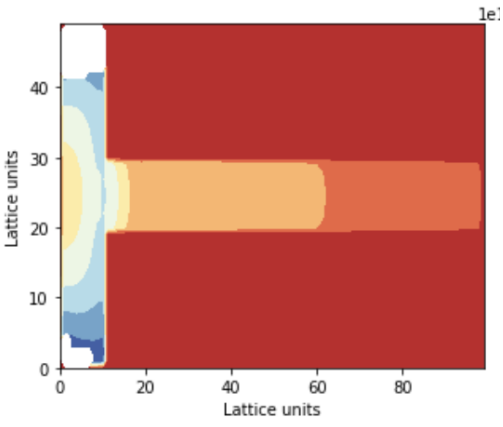
(e) Case e) after $t = 1.33\ s$, droplet flow.



(f) Case f) after $t = 7.98\ s$, parallel flow.



(g) Case g) after $t = 1.53\ s$, slug-droplet flow.



(h) Case h) model cannot handle inlet speeds.

Case a) shows that the model does not work when both fluids have the same inlet flow rate. The blue fluid intrudes into the inlet of the red fluid. This could be explained by the way the fluids are initialized within the T-shaped channel. Flow rate ratios larger than 4 showed clear flow patterns, as can be seen for cases b) and c). Case b) shows slug flow for low flow rates and a ratio of 5, while case c) shows droplet flow for higher flow rates with the same ratio. Cases d) and e) both have a flow rate ratio of 10, with case d) again showing slug flow for lower flow rates and and case e) again showing droplet flow for higher flow rates. Cases f), g) and h) show the highest flow rate ratios. Case f) shows that a high flow rate ratio results in parallel flow for an water inlet flow rate that previously resulted in slug flow. Case g) shows slug-droplet flow with small droplets of toluene going into the water inlet. Case h) shows that the model cannot handle inlet speeds above $u = 0.1$. This can be explained by the fact that u cannot be larger than the sound speed c_s^2, which in this case is $\frac{1}{3}$. Having a inlet flow rate that is $\frac{1}{3}$ the speed of sound is obviously physically improbable. Higher inlet flow rates could be achieved by increasing the characteristic length L or decreasing the kinematic viscosity ν and therefore τ in the lattice domain.

Case e) and g) show larger densities in between the droplets and slugs, showing that speeds above $u = 0.03$ already cause irregularities in the model. This might have to do with the way surface tension σ is incorporated into the model. For stability and simplicity, the variable A, which incorporates surface tension into the model, is kept constant at $A = 0.0001$. However, this is physically inaccurate. A is related to the kinematic viscosity through relations 42 and 43. Since surface tension is a constant force, A has to vary with ν, since ν changes at the interface between the two fluids. Keeping A constant therefore means that the surface tension changes at the interface, which is physically inaccurate. Case f) clearly shows that water is taken as fully wetting and toluene as fully non-wetting in these simulations, since the red fluid does not touch the lower wall of the channel. This is also physically inaccurate, since water is not fully wetting and toluene is not fully non-wetting.

These results show that the color-gradient RK model is able to simulate multiphase flow through a T-shaped microchannel. The simulations show that model parameters such as the relaxation time τ, which cannot be too close to 0.5, interface parameters A and β and sound speed c_s^2 have a large influence on the stability and accuracy of the model.

4.3 Comparison with experimental results

In experimental research, Kashid et al found that increasing the inlet flow rates of both fluids increases the tendency towards parallel flow. Increasing the flow rate ratio of water and toluene increases the tendency towards deformed interface flow. Their results are shown in Figure 14.

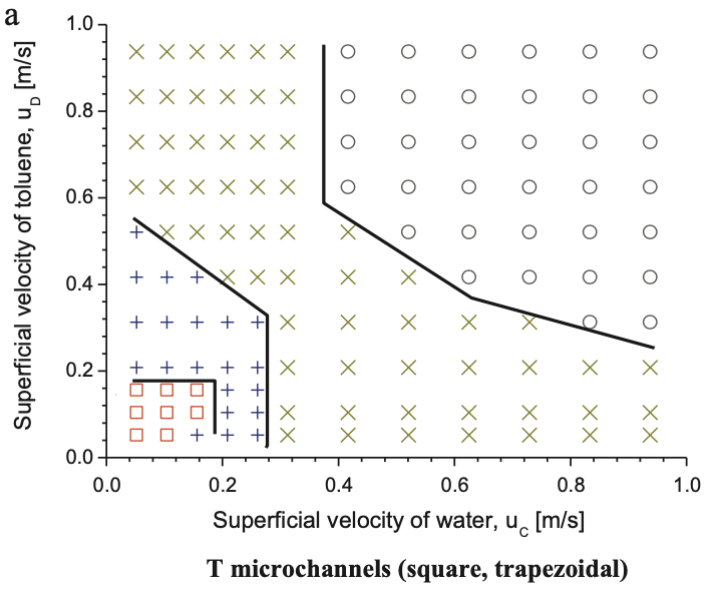


Figure 14: Findings of Kashid et al [7] on the effect of flow rate on flow patterns inside a T-shaped channel. (□) represent slug flow, (+) represents slug-droplet flow, (×) represents deformed interface flow and (◯) represents parallel flow.

From this data, it can be seen that in the low flow rates regime, slug flow can be observed, followed by droplet flow, deformed interface flow and parallel flow when moving towards higher flow rates. A similar plot of all simulated cases in this research is shown in Figure 15.

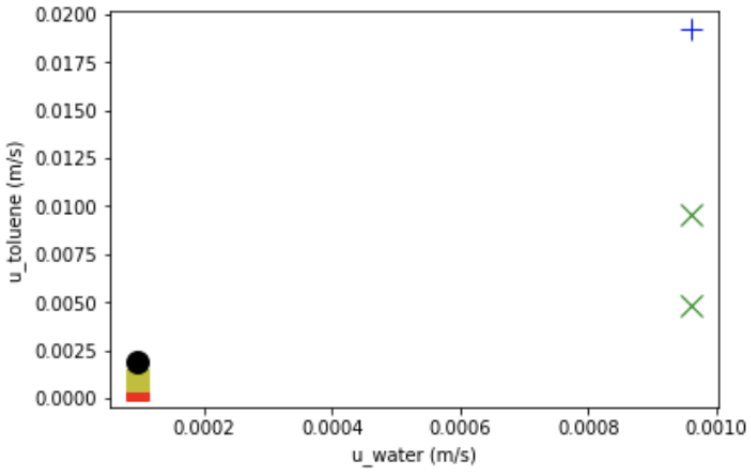


Figure 15: Plot of flow patterns observed at each case for the corresponding inlet flow rates. (□) represents slug flow, (+) represents droplet flow, (∘) represents parallel flow and (×) represents slug-droplet flow.

When comparing the results of the simulation with the experimental results, there are some key differences. For both cases, lower inlet flow rates tend towards slug flow, while higher flow rates show droplet and parallel flow. But during simulations, no deformed interface flow was observed. For high flow rate ratios, parallel and slug-droplet flow was observed in simulations, while the experimental results show that this increases the tendency towards deformed interface flow. The biggest difference is the inlet flow rates at which flow patterns are observed. As stated in section 4.2, the model was not able to simulate any flow patterns for equal flow rates and the model became unstable for flow rates higher than $u = 0.03\ m/s$. In this flow rate regime, Kashid et al observed only slug flow as can be seen in Figure 14, while in the simulations, most flow patterns were already observed for lower flow rates. This is most likely due to the inaccurate ways of implementing contact angle and surface tension as mentioned in section 4.2.

When a fluid wets a solid, it resists flow, since fluid molecules stick to the solid surface. Since toluene is fully non-wetting in the model used, it feels no resistance to flow from the channel walls. This might be the reason why parallel flow is already observed at relatively small inlet flow rates in the simulations.

Slug flow can be observed when both inlet flow rates are low, because interfacial forces dominate inertial forces [7]. When flow rates are increased, inertial forces become larger and interfacial forces cannot maintain the shape of slugs, 'knocking' toluene down into droplets [3]. At even higher flow rates, inertial forces dominate interfacial forces, so that the two faces can flow next to each other without generating slugs or droplets. Interfacial forces therefore are key in the formation of flow patterns. Surface tension, being an interfacial force, therefore needs to be accurately incorporated into the model to simulate different types of flow patterns. Since the model used in this research varies surface tension at the interface, the model does not accurately incorporate surface tension. This is also a reason why the results of the simulations do not match experimental results.

The results of color-gradient RK model used in this research therefore do not match experimental results.

5 Conclusion

This research focuses on analysing flow patterns of two immiscible fluids through a T-shaped microchannel by simulating the two-phase flow using the color-gradient Rothman-Keller Lattice Boltzmann model. The research aims to investigate whether the color-gradient RK model is able to simulate different types of flow patterns inside the T-shaped channel, what model parameters influence the accuracy, stability and type of flow pattern observed inside the channel and if the results of the simulations match the results of previous experimental research.

Wall, inlet, outlet and wetting boundary conditions were imposed to best simulate multiphase flow through the microchannel. Fluid properties and channel dimensions were adopted from previous experimental research to compare the results of the simulations to experimental results. Different flow patterns were obtained by varying inlet flow rates of the two fluids. Simulations were done using water and toluene as the two fluids and a channel width of 400 μm to replicate the experimental set-up of Kashid et al.

The results of the simulations show that for low flow rates u and low flow rate ratios, slug flow is observed inside the T-shaped channel. For medium flow rates and flow rate ratios, droplet flow was observed. For high flow rates and flow rate ratios, parallel and slug-droplet flow was observed. This shows that the color-gradient RK model is capable of simulating flow patterns of two-phase flow in a T-shaped microchannel.

When comparing the results of the simulations with the experimental results of Kashid et al, there are some key differences. The flow rate regimes where slug, droplet and parallel flow were observed in the research of Kashid et al were an order of magnitude higher. Deformed interface flow, a flow pattern observed for multiple flow rates in the experimental research, was not observed in the simulations. The model was not capable of simulating flow rates higher than $u = 0.03\ m/s$ and flow rate ratios smaller than 5, while Kashid et al were able to get results outside these values. This shows that the results of the model do not match the results of experimental research done by Kashid et al. This is largely due to the way that surface tension and contact angle are incorporated into the model. In the model, water is taken to be fully wetting, while toluene is taken to be fully non-wetting. Surface tension in the model varies at the interface, while this should be a constant force. Both these traits are physically inaccurate.

The simulations also showed that model parameters relaxation time τ, surface tension A, interface thickness β, sound speed c_s^2 and contact angle θ affected the stability and accuracy of the model. The model was unstable for τ too close to 0.5 and inlet flow rates u too close to the sound speed c_s^2. The model was inaccurate for constant values for A, values outside of 0.5 - 0.7 for β and contact angles θ of 0 and 180 degrees for toluene and water respectively.

6 Recommendations

For further research, certain aspects of the model used in this research could be improved.

First, this research considered one fluid to be fully wetting, a contact angle of 180 degrees, and the other fluid to be fully non-wetting, a contact angle of 0 degrees, for simplicity. This is unrealistic, since no fluid is fully wetting or fully non-wetting. Ding and Spelt [17] proposed a wetting boundary condition which Liu et al [18] incorporated into a multi relaxation time (MRT) color-fluid model. A better wetting boundary condition could be imposed to improve the accuracy of the model.

Second, the way surface tension is defined in the model could be improved. For stability and simplicity, the variable A, which incorporates surface tension into the model, is kept constant at $A = 0.0001$. However, this is physically inaccurate. A is related to viscosity through relations 42 and 43. Since surface tension is a constant force, A has to vary with ν, since ν changes at the interface between the two fluids. Keeping A constant therefore means that the surface tension changes at the interface. This is physically inaccurate. Liu et al [18] adopted a scheme where surface tension could be implemented directly into the model. Adopting a better method to incorporate surface tension will improve the accuracy of the model.

Third, to achieve more data points at higher inlet flow rates and therefore higher Reynolds and capillary numbers, a smaller length conversion factor Δx should be used. To achieve higher Reynolds numbers, one could increase the inlet flow rate u, increase the characteristic length L or decrease ν in the lattice domain. However, decreasing ν means decreasing the value for τ, but values too close to 0.5 affect the model's stability. Increasing inlet flow rate u is also not endless, since u cannot be bigger than the sound speed c_s^2, which is $\frac{1}{3}$. Increasing the value the value of the characteristic length L remains as the last option, which means increasing the computational domain and therefore the computational power needed to run the simulations. This needs to be taken into account for further research. More data points could also indicate at what inlet flow rates certain flow patterns start to shift to other flow patterns, indicated by the black lines in Figure 14 by Kashid et al.

Lastly, the model could also be adopted to simulate the effects of channel geometries such as channel width, inlet width and channel length and fluid properties such as different density and viscosity ratios on the flow patterns observed inside the channel. This could give a more detailed understanding into what parameters and fluid properties lead to certain flow patterns.

References

N. Assmann, A. Ladoxz, and P. Rudlof von Rohr. Continuous micro liquid-liquid extraction. *Chemical Engineering And Technology*, pages 921–936, 2013.

J. Qian, X. Li, Z. Wu, Z. Jin, and B. Sunden. A comprehensive review on liquid-liquid two-phase flow in a microchannel: flow pattern and mass transfer. *Microfluidics and Nanofluidics*, 2019.

M. Darekar, K. Singh, S. Mukhopadhyay, and K. Shenoy. Liquid-liquid two-phase flow patterns in y-junction microchannels. *Industrial & Engineering Chemistry Research*, 56(42):12215–12226, 2017.

L. Shui, J. Eijkel, and A. van den Berg. Multiphase flow in microfluidic systems - control and applications of droplets and interfaces. *Advances in colloid and interface science*, pages 35–49, 2007.

P. Jahormi, J. Karimi-Sabet, and Y. Amini. Ion-pair extraction-reaction of calcium using y-shaped microfluidic junctions: An optimized separation approach. *Chemical Engineering Journal*, pages 2603–2615, 2017.

Y. Zhao, G. Chen, and Q. Yan. Liquid-liquid two-phase mass transfer in t-junction microchannels. *AIChE Journal*, 53(12):3042–3053, 2007.

M. Kashid and L. Kiwi-Minsker. Quantitative prediction of flow patterns in liquid-liquid flow in micro-capillaries. *Chemical Engineering and Processing: Process Intensification*, pages 972–978, 2011.

J. Huang, F. Xiao, and X. Yin. Lattice boltzmann simulation of pressure-driven two-phase flows in capillary tube and porous medium. *Computers and Fluids*, pages 134–145, 2017.

T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. Viggen. *The Lattice Boltzmann Method, principles and practice*. Springer, 2017.

H. Huang, M. Sukop, and X. Lu. *Multiphase Lattice Boltzmann Methods, Theory and Application*. John Wiley & Sons, 2015.

D. Rothman and J. Keller. Immiscible cellular-automaton fluids. *Journal of Statistical Physics*, pages 1119–1127, 1988.

X. Shan and H. Chen. Lattice boltzmann model for simulating flows with multiple phases and components. *Physical Review E*, pages 1815–1820, 1993.

M.R. Swift, E. Orlandini, and J.M. Yeomans. Lattice boltzmann simulations of liquid-gas and binary fluid systems. *Physical Review E*, pages 5041–5052, 1996.

T. Reis and T. N. Phillips. Lattice boltzmann model for simulating immiscible two-phase flows. *J. Phys. A: Math. Gen.*, pages 1–23.

A.J.C. Ladd. Numerical simulations of particulate suspensions via a discretized boltzmann equation. part 1. theoretical foundation. *Journal of Fluid Mechanics*, pages 285–309, 1994.

Q. Lou, Z. Guo, and B. Shi. Evaluation of outlfow boundary conditions for two-phase lattice boltzmann equation. pages 063301–1 – 063301–16, 2013.

H. Ding and P. Spelt. Wetting condition in diffuse interface simulations of contact line motion. *Physical Review E*, 2007.

H. Liu, Y. Ju, N. Wang, G. Xi, and Y. Zhang. Lattice boltzmann modeling of contact angle and its hysteresis in two-phase flow with large viscosity difference. pages 1–18, 2015.

7 Appendix

7.1 Unit conversion of model parameters

The physical parameters of the experimental set-up are converted to Lattice Boltzmann units as follows.

First, a resolution for the channel width $L = 400\ \mu m$ is chosen. L^* is chosen to be 10 lu. The spatial resolution Δx is therefore $\Delta x = 40\ \mu m$. Next, the relaxation time τ^* is chosen. Since τ^* is related to the kinematic viscosity ν^* through equation 49:

$$\nu^* = \frac{1}{3}(\tau^* - 0.5) \tag{49}$$

The dynamic viscosity ratio of water and toluene is 1.69, giving a kinematic viscosity ratio of 1.47. This ratio is the same in the lattice domain, so if τ^*_{water} is chosen to be 0.75, $\tau^*_{toluene} = 0.67$. The time resolution can be calculated through equation 50 [9]:

$$\Delta t = c_s^{*2}(\tau^* - \frac{1}{2})\frac{\Delta x^2}{\nu} \tag{50}$$

This gives a time resolution $\Delta t = 1.33 \cdot 10^{-4}$ s. The conversion factor for u, $C_u = \frac{\Delta x}{\Delta t} = 0.3$ m/s.

7.2 Python script

```
# Color-gradient Rothman-Keller Model

from sympy.solvers import solve
from sympy import Symbol
from numba import njit, jit, float64, int64
import numpy as np
import matplotlib.pyplot as plt
import scipy as sci
import time

start = time.time()
# Grid domain
lx = 50
ly = 100
width = 10 # channel and inlet width
in_length = int((lx-width)/2)

# Velocity, density and particle density arrays
u_x = np.zeros([lx,ly])
u_y = np.zeros([lx,ly])
rho = np.zeros([lx,ly])
ff = np.zeros([9,lx,ly])
f_b = np.zeros([9,lx,ly])
f_r = np.zeros([9,lx,ly])
Fx = np.zeros([lx,ly])
Fy = np.zeros([lx,ly])
p = np.zeros([lx,ly])

# D2Q9 Model
ex = np.array([0.0, 1.0, 0.0, -1.0, 0.0, 1.0, -1.0, -1.0, 1.0])
ey = np.array([0.0, 0.0, 1.0, 0.0, -1.0, 1.0, 1.0, -1.0, -1.0])
cc = 1
c_squ = cc**2/3
wi = np.array([4/9, 1/9, 1/9, 1/9, 1/9, 1/36, 1/36, 1/36, 1/36])
Bi = np.array([-4/27, 2/27, 2/27, 2/27, 2/27, 5/108, 5/108, 5/108, 5/108])
rsqu = np.array([0.0, 1.0, 1.0, 1.0, 1.0, np.sqrt(2), np.sqrt(2), np.sqrt(2), np.sqrt(2)])
```

```
# Parameters
beta = 0.7 # Parameter for thickness of interface
A = 0.0001 # Parameter to adjust the strength of the interface tension
df = 0.000000015 # Body force applied to the fluid
alpha_r = 0.42
alpha_b = 3/9
rho_ri = 1 - alpha_b
rho_bi = 1 - alpha_r
c_squ_r = 3*(1-alpha_r)/5
c_squ_b = 3*(1-alpha_b)/5
taur = 0.75
taub = 0.67

# Inlet speeds
u_w1 = 0.0016 # Red fluid
u_w2 = -0.00032 # Blue fluid

# Making obstacle nodes; obst = 0 is fluid node, obst >= 1 is solid node
obst = np.zeros([lx,ly])
obst[:,0]=1
obst[0:int((lx-width)/2), width+1]=4
obst[int((lx+width)/2):lx, width+1]=4
obst[in_length-1,width+1:ly]=5
obst[in_length+width,width+1:ly]=6
obst[0:int((lx-width)/2)-1, width+2:ly]=-1
obst[int((lx+width)/2)+1:lx, width+2:ly]=-1
obst[0,1:width+1]=2
obst[lx-1,1:width+1]=3
obst[in_length:in_length+width, ly-1]=7
obst[in_length-1,width+1]=8
obst[in_length+width,width+1]=9


# Find equilibrium distribution function
@njit
def feq(rho_r,rho_b,u_x,u_y,alpha_r,alpha_b,wi,c_squ,ex,ey):
    un= np.zeros((9,lx,ly))
    f_eqr=np.zeros((9,lx,ly))
    f_eqb=np.zeros((9,lx,ly))
    for i in range(0,9,1):
        un[i,:,:]=ex[i]*u_x+ey[i]*u_y
        if (i==0):
            f_eqr[i,:,:]=wi[i]*rho_r*(un[i,:,:]/c_squ+ np.power(un[i,:,:],2)*4.5 -
                    np.power(un[i,:,:],2)*1.5) +rho_r*alpha_r
            f_eqb[i,:,:]=wi[i]*rho_b*(un[i,:,:]/c_squ+ np.power(un[i,:,:],2)*4.5 -
                     np.power(un[i,:,:],2)*1.5) +rho_b*alpha_b

        elif (i>0 and i<5):
            f_eqr[i,:,:]=wi[i]*rho_r*(un[i,:,:]*3+ np.power(un[i,:,:],2)*4.5 -
                                np.power(un[i,:,:],2)*1.5) +rho_r*(1-alpha_r)/5
            f_eqb[i,:,:]=wi[i]*rho_b*(un[i,:,:]*3+ np.power(un[i,:,:],2)*4.5 -
                                 np.power(un[i,:,:],2)*1.5) +rho_b*(1-alpha_b)/5
        else:
            f_eqr[i,:,:]=wi[i]*rho_r*(un[i,:,:]*3+ np.power(un[i,:,:],2)*4.5 -
                                np.power(un[i,:,:],2)*1.5) +rho_r*(1-alpha_r)/20
            f_eqb[i,:,:]=wi[i]*rho_b*(un[i,:,:]*3+ np.power(un[i,:,:],2)*4.5 -
                                np.power(un[i,:,:],2)*1.5) +rho_b*(1-alpha_b)/20
```

```
    return f_eqr,f_eqb


# Find initial density
rho_r = np.zeros([lx,ly])
rho_b = np.zeros([lx,ly])
for x in range(int((lx-width)/2)-1):
    rho_b[x,:] = rho_bi
for x in range(int((lx-width)/2)-1, lx):
    rho_r[x,:] = rho_ri
rho = rho_r + rho_b

for x in range(lx):
    for y in range(ly):
        if obst[x,y] == -1:
            rho_r[x,y] = 0
            rho_b[x,y] = 0
            rho[x,y] = 0
        elif obst[x,y] == 5 and obst[x,y] == 8:
            rho_r[x,y] = rho_ri
            rho_b[x,y] = 0

f_r = feq(rho_r,rho_b,u_x,u_y,alpha_r,alpha_b,wi,c_squ,ex,ey)[0]
f_b = feq(rho_r,rho_b,u_x,u_y,alpha_r,alpha_b,wi,c_squ,ex,ey)[1]
ff = f_r + f_b



# Redistribute function
@njit
def redistribute(obst,rho_r,rho_b,rho,Fx,Fy,f_r,f_b,ff,ex,ey,lx,ly,wi,rsqu,A,Bi,beta):

            fm= np.zeros((lx,ly))
            for i in range(lx):
                for j in range(ly):
                    if obst[i,j] == 0 :
                        jn= j%(ly-1)+1
                        ie= i%(lx-1)+1
                        js= ly-1-(ly-j)%(ly)
                        iw= lx-1-(lx-i)%(lx)
                        Fx[i,j]= ex[1]*(rho_r[ie,j]-rho_b[ie,j])+ex[5]*(rho_r[ie,jn]-rho_b
                        ex[8]*(rho_r[ie,js]-rho_b[ie,js])+ex[3]*(rho_r[iw,j]-rho_b[iw,j])
                        ex[6]*(rho_r[iw,jn]-rho_b[iw,jn])+ex[7]*(rho_r[iw,js]-rho_b[iw,js]
                        Fy[i,j]= ey[2]*(rho_r[i,jn]-rho_b[i,jn])+ ey[5]*(rho_r[ie,jn]-rho_
                        ey[6]*(rho_r[iw,jn]-rho_b[iw,jn])+ey[4]*(rho_r[i,js]-rho_b[i,js])+
                        ey[7]*(rho_r[iw,js]-rho_b[iw,js])+ey[8]*(rho_r[ie,js]-rho_b[ie,js]
                        fm[i,j]= np.power((np.power(Fx[i,j],2)+np.power(Fy[i,j],2)),0.5)
                        cosfai= np.zeros((9,lx,ly))
                        eqf= np.zeros((9,lx,ly))
                        for k in range(0,9):
                            # Cases for denominator=0
                            if (fm[i,j]<10**-8 and fm[i,j]>=0):
                                f_r[k,i,j]= ff[k,i,j]*rho_r[i,j]/rho[i,j]
                                f_b[k,i,j]= ff[k,i,j]*rho_b[i,j]/rho[i,j]
                            else:
                                if (k==0):
                                    ff[k,i,j]= ff[k,i,j]+A*fm[i,j]*(-Bi[k]);
                                    f_r[k,i,j]= rho_r[i,j]*ff[k,i,j]/rho[i,j];
```

```
                                    f_b[k,i,j]= rho_b[i,j]*ff[k,i,j]/rho[i,j];
                                else:
                                    cosfai[k,i,j]= (ex[k]*Fx[i,j]+ ey[k]*Fy[i,j])/(rsqu[k]
                                    ff[k,i,j]= ff[k,i,j]+A*fm[i,j]*(wi[k]*np.power(cosfai[
                                    tem= rho_r[i,j]*rho_b[i,j]/(np.power(rho[i,j],2));
                                # Redistributed f's
                                    eqf[k,i,j]= wi[k]*rho[i,j];
                                    f_r[k,i,j]= rho_r[i,j]*ff[k,i,j]/rho[i,j]+ beta*tem*eq
                                    f_b[k,i,j]= rho_b[i,j]*ff[k,i,j]/rho[i,j]-beta*tem*eqf

            return f_r,f_b


# Bounce back boundary condition non-slip wall
@njit
def bounceback(ff, ffc, wi, rho, ex, u_w1, u_w2, c_squ, u_y):
    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 1 and x == 0:
                ff[5,x+1,y+1] = ffc[7,x+1,y+1]
            elif obst[x,y] == 1 and x>0 and x<lx-1:
                ff[2,x,y+1] = ffc[4,x,y+1]
                ff[5,x+1,y+1] = ffc[7,x+1,y+1]
                ff[6,x-1,y+1] = ffc[8,x-1,y+1]
            elif obst[x,y] == 1 and x == lx-1:
                ff[6,x-1,y+1] = ffc[8,x-1,y+1]
            elif obst[x,y] == 2:
                ff[1,x+1,y] == ffc[3,x+1,y] - 2*wi[3]*rho[x+1,y]*ex[3]*u_w1/c_squ
                ff[5,x+1,y+1] = ffc[7,x+1,y+1] - 2*wi[7]*rho[x+1,y+1]*ex[7]*u_w1/c_squ
                ff[8,x+1,y-1] = ffc[6,x+1,y-1] - 2*wi[6]*rho[x+1,y-1]*ex[6]*u_w1/c_squ
            elif obst[x,y] == 3:
                ff[3,x-1,y] = ffc[1,x-1,y] - 2*wi[1]*rho[x-1,y]*ex[1]*u_w2/c_squ
                ff[6,x-1,y+1] = ffc[8,x-1,y+1] - 2*wi[8]*rho[x-1,y+1]*ex[8]*u_w2/c_squ
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]- 2*wi[5]*rho[x-1,y-1]*ex[5]*u_w2/c_squ
            elif obst[x,y] == 4 and x == 0:
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]
            elif obst[x,y] == 4 and x>0 and x<lx-1:
                ff[4,x,y-1] = ffc[2,x,y-1]
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]
            elif obst[x,y] == 4 and x == lx-1:
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
            elif obst[x,y] == 5 and y < ly-1:
                ff[1,x+1,y] = ffc[3,x+1,y]
                ff[5,x+1,y+1] = ffc[7,x+1,y+1]
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]
            elif obst[x,y] == 5 and y == ly-1:
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]
            elif obst[x,y] == 6 and y < ly-1:
                ff[3,x-1,y] = ffc[1,x-1,y]
                ff[6,x-1,y+1] = ffc[8,x-1,y+1]
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
            elif obst[x,y] == 6 and y == ly-1:
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
            elif obst[x,y] == 7:
                ff[4,x,y-1] = (ff[4,x,y-1] + ff[4,x,y-2]*np.mean(u_y[in_length:in_length+w
                ff[7,x,y-1] = (ff[7,x,y-1] + ff[7,x,y-2]*np.mean(u_y[in_length:in_length+w
                ff[8,x,y-1] = (ff[8,x,y-1] + ff[8,x,y-2]*np.mean(u_y[in_length:in_length+w
```

```
            elif obst[x,y] == 8:
                ff[1,x+1,y] = ffc[3,x+1,y]
                ff[4,x,y-1] = ffc[2,x,y-1]
                ff[5,x+1,y+1] = ffc[7,x+1,y+1]
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]
            elif obst[x,y] == 9:
                ff[3,x-1,y] = ffc[1,x-1,y]
                ff[4,x,y-1] = ffc[2,x,y-1]
                ff[6,x-1,y+1] = ffc[8,x-1,y+1]
                ff[7,x-1,y-1] = ffc[5,x-1,y-1]
                ff[8,x+1,y-1] = ffc[6,x+1,y-1]

    return ff


# Streaming step
@njit
def stream(ff, width, in_length):
    f_hlp = np.copy(ff)
    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 1 and x == 0:
                f_hlp[1,x+1,y]=ff[1,x,y]
                f_hlp[2,x,y+1]=ff[2,x,y]
                f_hlp[5,x+1,y+1]=ff[5,x,y]
            elif obst[x,y] == 1 and x>0 and x<lx-1:
                f_hlp[1,x+1,y]=ff[1,x,y]
                f_hlp[2,x,y+1]=ff[2,x,y]
                f_hlp[5,x+1,y+1]=ff[5,x,y]
                f_hlp[3,x-1,y]=ff[3,x,y]
                f_hlp[6,x-1,y+1]=ff[6,x,y]
            elif obst[x,y] == 1 and x == lx-1:
                f_hlp[2,x,y+1]=ff[2,x,y]
                f_hlp[3,x-1,y]=ff[3,x,y]
                f_hlp[6,x-1,y+1]=ff[6,x,y]
            elif obst[x,y] == 2:
                f_hlp[1,x+1,y]=ff[1,x,y]
                f_hlp[2,x,y+1]=ff[2,x,y]
                f_hlp[5,x+1,y+1]=ff[5,x,y]
                f_hlp[4,x,y-1]=ff[4,x,y]
                f_hlp[8,x+1,y-1]=ff[8,x,y]
            elif obst[x,y] == 3:
                f_hlp[2,x,y+1]=ff[2,x,y]
                f_hlp[3,x-1,y]=ff[3,x,y]
                f_hlp[4,x,y-1]=ff[4,x,y]
                f_hlp[6,x-1,y+1]=ff[6,x,y]
                f_hlp[7,x-1,y-1]=ff[7,x,y]
            elif obst[x,y] == 4 and x == 0:
                f_hlp[1,x+1,y]=ff[1,x,y]
                f_hlp[4,x,y-1]=ff[4,x,y]
                f_hlp[8,x+1,y-1]=ff[8,x,y]
            elif obst[x,y] == 4 and x>0 and x<lx-1:
                f_hlp[1,x+1,y]=ff[1,x,y]
                f_hlp[3,x-1,y]=ff[3,x,y]
                f_hlp[4,x,y-1]=ff[4,x,y]
                f_hlp[7,x-1,y-1]=ff[7,x,y]
                f_hlp[8,x+1,y-1]=ff[8,x,y]
```

```
elif obst[x,y] == 4 and x == lx-1:
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
elif obst[x,y] == 5 and y < ly-1:
    f_hlp[1,x+1,y]=ff[1,x,y]
    f_hlp[2,x,y+1]=ff[2,x,y]
    f_hlp[5,x+1,y+1]=ff[5,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[8,x+1,y-1]=ff[8,x,y]
elif obst[x,y] == 5 and y == ly-1:
    f_hlp[1,x+1,y]=ff[1,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[8,x+1,y-1]=ff[8,x,y]
elif obst[x,y] == 6 and y < ly-1:
    f_hlp[2,x,y+1]=ff[2,x,y]
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[6,x-1,y+1]=ff[6,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
elif obst[x,y] == 6 and y == ly-1:
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
elif obst[x,y] == 7:
    f_hlp[1,x+1,y]=ff[1,x,y]
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
    f_hlp[8,x+1,y-1]=ff[8,x,y]
elif obst[x,y] == 8:
    f_hlp[1,x+1,y]=ff[1,x,y]
    f_hlp[2,x,y+1]=ff[2,x,y]
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[5,x+1,y+1]=ff[5,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
    f_hlp[8,x+1,y-1]=ff[8,x,y]
elif obst[x,y] == 9:
    f_hlp[1,x+1,y]=ff[1,x,y]
    f_hlp[2,x,y+1]=ff[2,x,y]
    f_hlp[3,x-1,y]=ff[3,x,y]
    f_hlp[4,x,y-1]=ff[4,x,y]
    f_hlp[6,x-1,y+1]=ff[6,x,y]
    f_hlp[7,x-1,y-1]=ff[7,x,y]
    f_hlp[8,x+1,y-1]=ff[8,x,y]


else:
        y_n = y%(ly-1) + 1
        x_e = x%(lx-1) + 1
        y_s = ly - 1 - (ly - y)%(ly)
        x_w = lx - 1 - (lx - x)%(lx)
        if x == (lx-1):
            x_e = 0
        if y == (ly-1):
            y_n = 0
```

```
                    f_hlp[1,x_e,y  ] = ff[1,x,y]
                    f_hlp[2,x  ,y_n] = ff[2,x,y]
                    f_hlp[3,x_w,y  ] = ff[3,x,y]
                    f_hlp[4,x  ,y_s] = ff[4,x,y]
                    f_hlp[5,x_e,y_n] = ff[5,x,y]
                    f_hlp[6,x_w,y_n] = ff[6,x,y]
                    f_hlp[7,x_w,y_s] = ff[7,x,y]
                    f_hlp[8,x_e,y_s] = ff[8,x,y]

    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 0:
                ff[1,x,y] = f_hlp[1,x,y]
                ff[2,x,y] = f_hlp[2,x,y]
                ff[3,x,y] = f_hlp[3,x,y]
                ff[4,x,y] = f_hlp[4,x,y]
                ff[5,x,y] = f_hlp[5,x,y]
                ff[6,x,y] = f_hlp[6,x,y]
                ff[7,x,y] = f_hlp[7,x,y]
                ff[8,x,y] = f_hlp[8,x,y]

    return ff


# Get macro variables
@njit
def getuv(u_x, u_y, rho, rho_r, rho_b, f_r, f_b):
    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 0:
                rho_r[x,y] = f_r[0,x,y] + f_r[1,x,y] + f_r[2,x,y] + f_r[3,x,y] + f_r[4,x,y
                rho_b[x,y] = f_b[0,x,y] + f_b[1,x,y] + f_b[2,x,y] + f_b[3,x,y] + f_b[4,x,y
                rho[x,y] = rho_r[x,y] + rho_b[x,y]
                u_x[x,y] = (f_b[1,x,y] + f_b[5,x,y] + f_b[8,x,y] - f_b[3,x,y] - f_b[6,x,y]
                u_y[x,y] = (f_b[2,x,y] + f_b[5,x,y] + f_b[6,x,y] - f_b[4,x,y] - f_b[7,x,y]

            elif obst[x,y] == 1 and obst[x,y] > 3: % Wetting
                rho_r[x,y] = rho_ri
                rho_b[x,y] = 0
            elif obst[x,y] == 2:
                rho_r[x,y] = 0
                rho_b[x,y] = rho_bi
            elif obst[x,y] == 3:
                rho_r[x,y] = rho_ri
                rho_b[x,y] = 0
    return(rho_r, rho_b, rho, u_x, u_y)

# Collision step
@njit
def collision(u_x, u_y, rho_r, rho_b, f_r, f_b, ff, taur, taub, c_squ_r, c_squ_b, c_squ, ex


    [fequ1, fequ2] = feq(rho_r,rho_b,u_x,u_y,alpha_r,alpha_b,wi,c_squ,ex,ey)

    delta1 = 0.98
    alpha1 = 2*taur*taub/(taur + taub)
    beta1 = 2*(taur - alpha1)/delta1
    kappa1 = -beta1/(2*delta1)
```

```
    eta1 = 2*(alpha1 - taub)/delta1
    kxi1 = eta1/(2*delta1)
    pr = np.zeros((lx,ly))
    pb = np.zeros((lx,ly))
    cr = np.zeros((lx,ly))
    cb = np.zeros((lx,ly))
    par = np.zeros((lx,ly))
    pab = np.zeros((lx,ly))
    G0 = np.zeros((lx,ly))
    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 0:
                if y > 1 and y < (ly-1):
                    pr[x,y] = 0.5*(rho_r[x,y+1]-rho_r[x,y-1])
                    pb[x,y] = 0.5*(rho_b[x,y+1]-rho_b[x,y-1])

    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 0:
                if y > 1 and y < (ly-1):
                    par[x,y] = 0.5*(pr[x,y+1]*u_x[x,y+1] - pr[x,y-1]*u_x[x,y-1])
                    pab[x,y] = 0.5*(pb[x,y+1]*u_x[x,y+1] - pb[x,y-1]*u_x[x,y-1])

    for x in range(lx):
        for y in range(ly):
            if obst[x,y] == 0:
                fei = (rho_r[x,y] - rho_b[x,y])/(rho_r[x,y] + rho_b[x,y])
                if fei > delta1:
                    tau = taur
                elif fei > 0 and fei <= delta1:
                    tau = alpha1 + beta1*fei + kappa1*fei*fei
                elif fei <= 0 and fei >= -delta1:
                    tau = alpha1 + eta1*fei + kxi1*fei*fei
                elif fei < -delta1:
                    tau = taub
                G0[x,y] = df
                cr[x,y] = (taur - 0.5)*(1/3 - c_squ_r)*par[x,y]
                cb[x,y] = (taub - 0.5)*(1/3 - c_squ_b)*pab[x,y]
                for k in range(9):
                    f_r[k,x,y] = fequ1[k,x,y] + (1 - 1/tau)*(f_r[k,x,y] - fequ1[k,x,y]) #-
                    f_b[k,x,y] = fequ2[k,x,y] + (1 - 1/tau)*(f_b[k,x,y] - fequ2[k,x,y]) #-
                    ff[k,x,y] = f_r[k,x,y] + f_b[k,x,y] # wi[k]*ex[k]*G0[x,y]/c_squ

    return ff, f_r, f_b

import time
t_max = 100000
start = time.time()

[f_eqr0, f_eqb0] = feq(rho_r,rho_b,u_x,u_y,alpha_r,alpha_b,wi,c_squ,ex,ey)
f_rc = np.copy(f_eqr0)
f_bc = np.copy(f_eqb0)


# Model algorithm with plot function
for t in range(0, t_max):
    f_r = stream(f_r, width, in_length)
    f_b = stream(f_b, width, in_length)
```

```
f_r = bounceback(f_r, f_rc, wi, rho_r, ex, u_w1, u_w2, c_squ, u_y)
f_b = bounceback(f_b, f_bc, wi, rho_b, ex, u_w1, u_w2, c_squ, u_y)
[rho_r, rho_b, rho, u_x, u_y] = getuv(u_x, u_y, rho, rho_r, rho_b, f_r, f_b)
[ff, f_r, f_b] = collision(u_x, u_y, rho_r, rho_b, f_r, f_b, ff, taur, taub, c_squ_r,
[f_r, f_b] = redistribute(obst,rho_r,rho_b,rho,Fx,Fy,f_r,f_b,ff,ex,ey,lx,ly,wi,rsqu,A,
f_rc = np.copy(f_r)
f_bc = np.copy(f_b)
if t == 10:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('10')
    plt.show()
elif t == 10000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('10000')
    plt.show()
elif t == 12500:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('12500')
    plt.show()
elif t == 15000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('15000')
    plt.show()
elif t == 17500:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('17500')
    plt.show()
elif t == 20000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('20000')
    plt.show()
elif t == 25000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('25000')
    plt.show()
elif t == 30000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('30000')
    plt.show()
elif t == 40000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.xlabel('40000')
    plt.show()
elif t == 50000:
    plt.contourf(rho_b, cmap='RdBu')
    plt.colorbar()
    plt.show()
elif t == 75000:
    plt.contourf(rho_b, cmap='RdBu')
```

```
            plt.colorbar()
            plt.show()

end = time.time()
print('Time =', end - start)

plt.contourf(rho, cmap='RdBu')
plt.colorbar()
plt.show()
```