DRAFT USER'S GUIDE FOR LB2D_PRIME V2.0

December 4, 2012

Katie Bardsley, Mike Sukop, Danny Thorne

Earth and Environment, Florida International University

Department of Math, Physics and Computer Science (MPC), Georgetown College

1   INTRODUCTION

LB2D_Prime is a lattice Boltzmann (LB) code capable of simulating single and multiphase flows and solute/heat transport in geometrically complex domains. Flows and transport in porous media can also be computed. The program is complementary to the text "Lattice Boltzmann Modeling" by Sukop and Thorne (2007); more details on all of the types of simulations described here can be found in the book.

The code was written by a computer scientist (Dr. Thorne) and is relatively comprehensive and hence appears rather complicated. But in fact the code is quite easy to use. This document has been developed to assist potential users – including those whose primary computing environment is Microsoft Windows based – get the program running. The program's main capabilities may be best learned by initial exposure to examples. Users who progress beyond the simple example problems will find ample information in the documentation and of course in the source code itself. This manual is a work in progress.

1.1   REQUIRED SOFTWARE

To obtain a copy of the LB2D_Prime source code, simply find it on the internet by searching 'google code lb2d' and use the Command-line access "hg clone https://lb2d-prime-dev.googlecode.com/hg/ lb2d-prime-dev" in Cygwin (see below) or any Unix-based environment, to obtain the latest Developer's version – which is not likely to be fully documented. This will create subdirectories /doc, /in, /out, and /src. These contain the program documentation, input, output, and source code respectively. If for some reason you do not receive an out file be sure to make one in the LB2D_Prime folder; you do not need to put anything into it.

There are 2 software programs necessary to use LB2D_Prime in its current stage of development: Cygwin and a text editor. The editor can be as simple as Wordpad, or a program such as vi can be used under

Windows, Cygwin, or Linux. This guide will cover a few simple Cygwin and MATLAB commands. For more advanced applications however, more advanced understanding of those programs could be necessary.

Your Windows computer should come equipped with Wordpad. If you would like to use vi, it can be downloaded from the internet (e.g., http://www.vim.org/) for both Unix and Windows. Also available online is Cygwin; this can be found at http://www.cygwin.com. It may take a while to download this program. It is suggested to download and install the program straight from the internet. This may take three or more hours, but there can be complications if Cygwin is downloaded and then installed separately. When downloading/installing Cygwin there will be a list of all of the required files, please be sure to change the list from "Default" to "Install" by clicking on the "Default" next to "All". This will ensure that the compiler files will be correctly installed. If this is not done, Cygwin will be unable to compile LB2D_Prime.

1.2   OPTIONAL SOFTWARE

MATLAB is used to probe the simulation results more deeply. MATLAB must be purchased separately

2   GETTING STARTED

2.1   TEST SCRIPTS

A number of test scripts are provided to ensure the correct operation of the code. Scripts available include:

- Poiseuille_gravity_x.sh (17x17 slit domain, 15 lattice unit slit, flow in positive x direction)
- Poiseuille_gravity_y.sh
- Poiseuille_pressure_y.sh
- Taylor_pressure.sh

To run the test scripts from the Cygwin command line, type

```
sh scriptname.sh
```

As an example, he first script contains the following lines to copies the necessary files from the Mercurial archive, compile, and run the code:

```
pushd in
cp 17x17_x.bmp 17x17.bmp
popd
pushd src
cp flags_poiseuille_gravity_x.h flags.h
popd
make
./lb2d_prime ./in/params_poiseuille_gravity_x.in
```

2.2   COMPILING LB2D_PRIME

First you need to compile LB2D_Prime. To do this, open Cygwin, change the directory (by typing 'cd [path to your directory]') to the directory containing the above subdirectories. Once the path is correct, type 'make'; this will compile the program. Now the program is compiled and we can begin to run simulations.

Typing 'make mpi' will compile a parallel version of the code, provided that the compiler directories are all in the places specified in the make file.

2.3   RUNNING LB2D_PRIME

2.3.1   PRINCIPAL INPUT FILES

As mentioned in the introduction, the best way to learn LB is by examples. Before we begin, let us become familiar with the two principal input files of LB. One is the input geometry file ixj.bmp where i and j are the number of ***lattice units*** (lu) in the x and y directions. The program is provided with a few of these files which will be used in the example simulations though they are easy to create. The second major file is called params.in and contains most of the information needed for a simulation. Flags.h is a file located in the .src directory that enables pre-compilation selection of the type of simulation and defines some output options.

2.3.2   GEOMETRY INPUT FILE

The code reads the locations of obstacles to flow from bitmap files contained in the 'in' directory. Many types of input graphics files can be converted to this format and domains can easily be drawn with Microsoft Paint® for example. Currently LB2D_Prime supports 24-bit 'truecolor' BMPs with black corresponding to obstacles and white signifying open space. It is essential that only black and white pixels appear in the file.

The file naming convention is based on the lattice size and one pixel in the bmp corresponds to 1 lu. Thus, the input file for a domain 100 *lu* long in *x* by 25 *lu* in *y* would be '100x25.bmp'. When creating bitmap files for simulations, keep in mind the rule of thumb is that 4 or 5 lattice units in an open flow channel are the minimum for the simulation of realistic hydrodynamics (Succi, 2001).
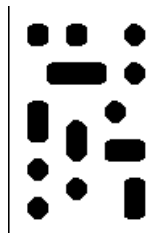

Figure 1: Example geometry file 96 by 144 pixels. Its name for input to the program is 96x144.bmp

2.3.3   PARAMETERS INPUT FILE

Also in the 'in' directory is the params.in file, which contains the main parameter values for the simulation. Note that many of the lines in the file may not be required for a given application and thus the number of lines a user needs to be concerned with and/or modify can be very small.

The detailed structure of the params.in file is similar to that indicated below (it can change slightly as new code releases become available; check the documentation corresponding to the release you are using for details). In the following example params.in file, the comments following '//' provide a brief description of the parameter on that line and what it is used for.

Distinct fluids are referred to both as components and as substances. Normally in conversation we talk about fluid components, but the parameters are indexed by "subs". The term "component" can be used to refer to a fluid component or a solute component, as well as the more broadly conventional mathematical usage of a vector component. The reader should therefore be wary of context.

The parameters that are arrays are stated with typical and/or suggestive index names between square brackets. For example, tau[0], tau[1] are written to suggest the fact that tau contains an array of values, one for each substance or fluid component.

A common mistake made when configuring a problem is to specify in the file "params.in" an integer number for a type double parameter or vice versa. This can cause mystifying symptoms. Be careful to include a decimal point in all type double parameter values in "params.in" regardless of whether there is a fractional part to the value. Likewise, do not include decimal points in type integer parameter values. Remember, if things have been going smoothly and then suddenly go bad in weird ways (sorry I can't recollect any specific examples) be sure to check this issue in "params.in".

The file "params.dat" is output by the code mostly as a sanity check: if values in "params.dat" do not match values in "params.in", the double versus integer parameter type issue may be the reason.

2.3.4   EXAMPLE OF PARAMS.IN FILE

LX	1000	// number of lattice nodes in x-direction
LY	600	// number of lattice nodes in y-direction
characteristic_length	31	// characteristic length used for computing Re only
NumFrames	100000	// number of frames to output
FrameRate	1000	// number of time steps per frame
tau[0]	0.55	// relaxation time for component 0

gforce_x[0]	0.0	// gravitational acceleration for component 0 in the x direction
gforce_y[0]	-0.000002222	// gravitational acceleration for component 0 in the y direction
end_grav[0]	0	// ending time step for gravity on fluid component 0
tau[1]	1.0	// relaxation time for component 1
gforce_x[1]	0.0	// gravitational acceleration for component 1 in the x direction
gforce_y[1]	0.0	// gravitational acceleration for component 1 in the y direction
end_grav[1]	0	// ending time step for gravity on fluid component 1
buoyancy	0	// toggles buoyancy effects due to the density of solute
incompressible	0	// toggles an incompressible lattice Boltzmann method
simple_diffusion	0	// toggles a simplified diffusion computation, by simplifying the
		// computation of f_{eq} for the solute component
rho_A	85.7042	// initial fluid density (see BCs)
rho_B	524.3905	// initial fluid density (see BCs)
rho_sigma	1.0	// initial density boundary condition for a solute component 1
rho_sigma_in	1.0	// 'in' density for a solute component 1
rho_sigma_out	0.0	// 'out' density for a solute component 1
u_sigma	0.0	// initial velocity (flux) boundary condition for a solute component 1
u_sigma_in	0.0	// 'in' velocity (flux) for a solute component 1
u_sigma_out	0.0	// 'out' velocity (flux) for a solute component 1
sigma_start	0	// time step at which boundary solute is activated
sigma_stop	9999999	// time step at which boundary solute is deactivated
sigma_btc_rate	100	// rate at which to accumulate temporal breakthrough curve
sigma_btc_spot	-1	// position in domain (from inflow) for measuring btc
rho_in	501.0	// 'in' density
rho_out	501.0	// 'out' density
ux_in	0.0	// 'in' velocity in x direction
ux_out	0.0	// 'out' velocity in the x direction
uy_in	0.0	// 'in' velocity in the y direction

uy_out	0.0	// 'out' velocity in the y direction
G	-120.0	// fluid-fluid or fluid-vapor interaction strength
Gads[0]	-327.79	// adsorption strength for component 0
Gads[1]	0.0	// adsorption strength for component 0
ns_flag	0	// 0/1 flag for Dardis & McCloksky porous medium
ns	0.1	// fraction of solids for Dardis & McCloksky porous medium
slice_x	500	// produces output along a slice through domain at this x-coordinate
slice_y	-10	// produces output along a slice through domain at this y-coordinate
ic_poisseuille	0	// 0/1 flag for for Poiseuille flow initial condition
bc_poisseuille	0	// 0/1 flag for Poiseuille flow velocity boundary condition
bc_slip_north	0	// 0/1 flag to toggle slip condition on north wall
bc_sigma_slip	0	// 0/1 flag for slip boundary condition for solute on side walls
pressure_n_in[0]	0	// 0/1/2 flag for north component 0 pressure BC = rho_in or file
pressure_s_in[0]	0	// 0/1 flag for south component 0 pressure BC = rho_in
pressure_n_out[0]	0	// 0/1 flag for north component 0 pressure BC = rho_out
pressure_s_out[0]	0	// 0/1 flag for south component 0 pressure BC = rho_out
velocity_n_in[0]	0	// 0/1 flag for north component 0 velocity BC = uy_in
velocity_s_in[0]	0	// 0/1 flag for south component 0 velocity BC = uy_in
velocity_n_out[0]	0	// 0/1 flag for north component 0 velocity BC = uy_out
velocity_s_out[0]	0	// 0/1 flag for south component 0 velocity BC = uy_out
pressure_e_in[0]	0	// 0/1 flag for east component 0 pressure BC = rho_in
pressure_w_in[0]	0	// 0/1/2 flag for west component 0 pressure BC = rho_in or file
pressure_e_out[0]	0	// 0/1 flag for east component 0 pressure BC = rho_out
pressure_w_out[0]	0	// 0/1 flag for west component 0 pressure BC = rho_out
velocity_e_in[0]	0	// 0/1 flag for east component 0 velocity BC = ux_in
velocity_w_in[0]	0	// 0/1 flag for west component 0 velocity BC = ux_in
velocity_e_out[0]	0	// 0/1 flag for east component 0 velocity BC = ux_out
velocity_w_out[0]	0	// 0/1 flag for east component 0 velocity BC = ux_out

pressure_n_in[1]	0	// 0/1 flag for north component 1 pressure BC = rho_in
pressure_s_in[1]	0	// 0/1 flag for south component 1 pressure BC = rho_in
pressure_n_out[1]	0	// 0/1 flag for north component 1 pressure BC = rho_out
pressure_s_out[1]	0	// 0/1 flag for south component 1 pressure BC = rho_out
velocity_n_in[1]	0	// 0/1 flag for north component 1 velocity BC = uy_in
velocity_s_in[1]	0	// 0/1 flag for south component 1 velocity BC = uy_in
velocity_n_out[1]	0	// 0/1 flag for north component 1 velocity BC = uy_out
velocity_s_out[1]	0	// 0/1 flag for south component 1 velocity BC = uy_out
pressure_e_in[1]	0	// 0/1 flag for east component 1 pressure BC = rho_in
pressure_w_in[1]	0	// 0/1 flag for west component 1 pressure BC = rho_in
pressure_e_out[1]	0	// 0/1 flag for east component 1 pressure BC = rho_out
pressure_w_out[1]	0	// 0/1 flag for west component 1 pressure BC = rho_out
velocity_e_in[1]	0	// 0/1 flag for east component 1 velocity BC = ux_in
velocity_w_in[1]	0	// 0/1 flag for west component 1 velocity BC = ux_in
velocity_e_out[1]	0	// 0/1 flag for east component 1 velocity BC = ux_out
velocity_w_out[1]	0	// 0/1 flag for east component 0 velocity BC = ux_out
constcon_n_in	0	// 0/1 flag for north constant concentration BC = rho_sigma_in
constcon_s_in	0	// 0/1 flag for south constant concentration BC = rho_sigma_in
constcon_n_out direction	0	// 0/1 flag for north constant concentration BC = rho_sigma_out
constcon_s_out direction	0	// 0/1 flag for south constant concentration BC = rho_sigma_out
constflx_n_in	0	// 0/1 flag for north constant solute flux BC = u_sigma_in
constflx_s_in	0	// 0/1 flag for south constant solute flux BC = u_sigma_in
constflx_n_out	0	// 0/1 flag for north constant solute flux BC = u_sigma_out
constflx_s_out	0	// 0/1 flag for south constant solute flux BC = u_sigma_out
constcon_e_in	0	// 0/1 flag for east constant concentration BC = rho_sigma_in
constcon_w_in	0	// 0/1 flag for west constant concentration BC = rho_sigma_in

constcon_e_out	0	// 0/1 flag for east constant concentration BC = rho_sigma_out
constcon_w_out	0	// 0/1 flag for east constant concentration BC = rho_sigma_out
constflx_e_in	0	// 0/1 flag for east constant solute flux BC = u_sigma_in
constflx_w_in	0	// 0/1 flag for south constant solute flux BC = u_sigma_in
constflx_e_out	0	// 0/1 flag for east constant solute flux BC = u_sigma_out
constflx_w_out	0	// 0/1 flag for west constant solute flux BC = u_sigma_out
zeroconcgrad_n	0	// 0/1 flag for north zero concentration gradient BC
zeroconcgrad_s	0	// 0/1 flag for south zero concentration gradient BC
zeroconcgrad_e	0	// 0/1 flag for east zero concentration gradient BC
zeroconcgrad_w	0	// 0/1 flag for west zero concentration gradient BC
zeroconcgrad_full	0	// 0/1 flag for zero concentration gradient
plot_scale_dynamic	0	// 0/1 flag for dynamic scaling of the plots
use_colormap	0	// see the colormap routines in lbio.c
initial_condition	8	// see below a listing/description of all initial conditions
x0	35.	// x-coordinate for the center of circle
y0	25.	// y-coordinate for the center of circle
r0	6.	// radius of circle
cut	-1.	// cut-off value for IC 7, see below
x1	-1.	// x-coordinate of lower left corner of a rectangle
x2	-1.	// y-coordinate of lower left corner of a rectangle
y1	-1.	// x-coordinate of upper right corner of a rectangle
y2	-1.	// y-coordinate of upper right corner of a rectangle
rel_x1	0.0	// relative x-coordinate of lower left corner of a rectangle
rel_x2	1.0	// relative x-coordinate of upper right corner of a rectangle
rel_y1	0.0	// relative y-coordinate of lower left corner of a rectangle
rel_y2	0.3	// relative y-coordinate of upper right corner of a rectangle
dump_rho	1	// 0/1 flag to toggle output of the macroscopic density
dump_u	0	// 0/1 flag to toggle output of the macroscopic velocity

| dump_force | 0 | // 0/1 flag to toggle output of the interaction and adsorption forces |
| dump_vor | 0 | // 0/1 flag to toggle output of vorticity |

2.3.5   INITIAL CONDITIONS

In all simulations you must input the initial conditions. Often, these can be very simple. In LB2D_Prime there are 13 initial conditions that can be used. Below is a list of the various settings. The leftmost column is the title of the condition, the center column is the number that would be entered into params.in, and the rightmost column is a brief description of the condition.

UNIFORM_RHO_A	1	//
UNIFORM_RHO_B	2	//
UNIFORM_RHO_IN	3	//
BUBBLE	4	// creates circle of density rho_A. When using this condition
		// the center and radius of the circle must be specified
DIAGONAL	5	//
2X2_CHECKERS	6	//
STATIC	7	// random with 'cut' equals fraction of component (subs) 1
RECTANGLE	8	// creates a rectangle of density rho_A:, when using this condition
		// specify the two corner coordinates, either relative or absolute.
DOT	9	//
WOLF_GLADROW_DIFFUSION	10	//
YIN_YANG	11	//
HYDROSTATIC	12	//
READ_FROM_FILE	99	// if none of the other conditions suit your needs, you may create
		// your own

2.3.6   TRANSIENT PRESSURE BOUNDARIES

Transient pressure values for substance 0 on the north and west boundaries may be specified externally in files named, for example, "pressure_n_in0.in", which contain a single column of pressure values, one for

each simulation time step. In this case, an integer "2" should follow the appropriate line in the params.in file as in: "pressure_n_in[0]   2".

If there are more timesteps than input values, the input values will be cycled. This is particularly useful for imposing temporally periodic boundary conditions to simulate something like tidal periods.

A convenient MATLAB code for generating the required file is

```
clear all
close all

a=(1 + 0.2*sin(0:2*pi/10000:2*pi))'

plot(a)

save pressure_n_in0.in a -ascii
```

This gives a mean density of 1 mu lu^{-3} with an amplitude of 0.2 mu lu^{-3} and a period of 10,000 time steps.

 More information on the variables in params.in can be found in the documentation folder of the program.

2.4   EXAMPLES

Now that you are familiar with the input we can try some examples. This section will explain how to do a few basic applications. For each example there is a simplified version of the params.in file that you can duplicate to run the respective examples. Please note that '…' means there are parameters included in params.in that do not need to be changed but must remain in the file. Also please forgive the repetitive nature of the example instructions; it was done this way so that you can begin at any example.

2.4.1   SINGLE COMPONENT, MULTIPHASE PHASE SEPARATION (SPINODAL DECOMPOSITION)

In this example we have one component, such as H_2O, but two phases, such as liquid water and water vapor. We start with an unstable initial density and watch as the liquid and vapor separate and ultimately attain their lowest energy state (circular drops or bubbles).

The first step is to open Cygwin and change to the directory 'cd' that contains LB2D_Prime. Next enter the src folder, rename the file flags_scmp.h to flags.h, and recompile the program by typing 'make' in Cygwin. Now enter the 'in' folder. A new .bmp file must be created here. In this example a blank area will suffice and all the domain boundaries will be periodic by default. The bigger the domain the longer the processing time, however the details of the simulation are easier to see. Try an area 100x100 lu, be sure to change the attributes of the .bmp to the correct size. Also in the 'in' folder, open the params.in (again you can use a variety of programs to edit this file). This example is a single component system so we will not be concerned with variables ending in [1].

Params.in

LX	100	
LY	100	
NumFrames	250	// this system will approach equilibrium if there are a large number of frames
FrameRate	100	// number of time steps per frame
tau[0]	1.0	
…		
rho_A	200.4	
…		
G	-120.0	// an attractive van der Waals-like force (negative interaction strength)
…		
slice_x	100	
slice_y	100	
…		
plot_scale_dynamic	1	
…		
initial_condition	7	// 'static' (random) initial condition
…		
cut	0.9	
…		
dump_rho	1	// outputs bitmaps (.bmp) of density
…		

Once all of these variables are entered, the program is ready to run. To do this, go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation, the output will be found in the out directory in .bmp form and should look similar to the figure below. You will notice that most of the drops are coalescing, however there are a few that disappear due to evaporation.
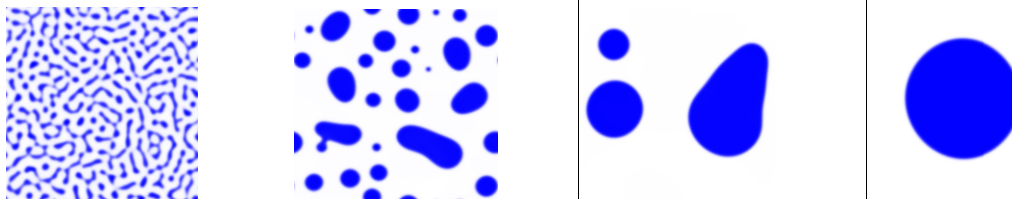

Figure 2: Example of a single component multiphase phase separation. Frames from left to right: 100, 800, 5,700, and 23,700 time steps in a 200x200 lu area.

2.4.2   SINGLE COMPONENT CONTACT ANGLE

In this example we again are only using one component. We begin with a circular initial condition and observe as it attaches itself to the wall with a 90° contact angle.

We start out the same as in the previous example by opening Cygwin and changing to the directory 'cd' containing LB2D_Prime. If you did not try the previous example you will have to rename the file flags_scmp.h to flags.h in the src folder. If you did the last example then you should still have this renamed. It is always a good idea to recompile the program, so again type 'make' into Cygwin. Now enter the 'in' folder, and create a new .bmp file. In this example we need a blank area with one wall. Try an area 100x100 lu, be sure to change the attributes of the .bmp to the correct size. Now open the params.in file, also in the 'in' folder. Again this example is a single component system so we will not be concerned with variables ending in [1].

Params.in

LX	100
LY	100
…	
NumFrames	100
FrameRate	100
tau[0]	1.0
…	
rho_A	200.4
…	
…	
rho_in	200.0
rho_out	200.0
…	
G	-120.0
Gads[0]	-187.16
…	
slice_x	50
slice_y	50
…	
plot_scale_dynamic	1
…	
initial_condition	4 // bubble
x0	25.0
y0	25.0
r0	10.0
…	
dump_rho	1
…	

Once all of these variables are entered, the program is ready to run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation. The output will be found in the out directory in .bmp form and should look similar to the figure below. The fuzzy halo around the solution is the transition to the vapor phase.
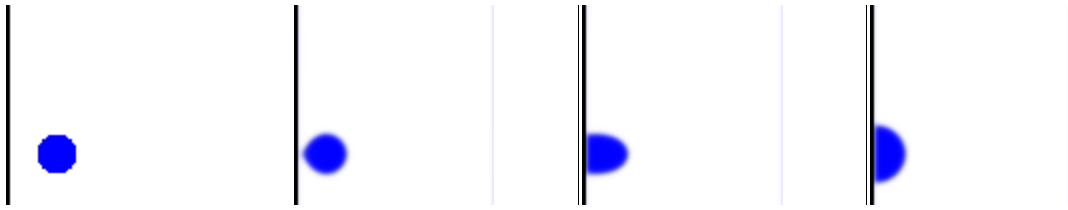
Figure 3: Example of a single component multiphase simulation with a contact angle of 90°. Frames from left to right: 0, 1200, 1400, and 5000 time steps in a 100x100 lu area.

2.4.3 MULTI-COMPONENT, MULTIPHASE INTERFACE RELAXATION

This example begins our journey into multi-component systems, such as oil and water. We start with a simple example of phase interface relaxation.

In Cygwin make sure that you are in the correct folder. Now enter the src folder, rename the file flags_mcmp.h to flags.h (you may want to rename your other flags.h back to flags_scmp.h), and recompile the program (type 'make' into Cygwin). Next enter the 'in' folder and create a new .bmp file. We are again using a blank area corresponding to a fully periodic domain. Let's use 100x100 lu. Be sure to change the attributes of the .bmp to the correct size. Also in the 'in' folder, open params.in. Unlike the previous examples the variables ending in [1] will now become important.

Params.in

LX	100
LY	100
…	
NumFrames	100
FrameRate	100
tau[0]	1.0
…	
tau[1]	1.0
…	
rho_A	1.0
rho_B	0.0
…	
rho_in	1.0
rho_out	1.0
…	
G	2.0
…	
slice_x	10
slice_y	10
…	
plot_scale_dynamic	1

```
…
initial_condition        8          // rectangle
…
x1                      -1.0        // -1 turns off absolute coordinates of rectangle
x2                      -1.0        // -1 turns off absolute coordinates of rectangle
y1                      -1.0        // -1 turns off absolute coordinates of rectangle
y2                      -1.0        // -1 turns off absolute coordinates of rectangle
rel_x1                   0.25       // gives relative coordinates of rectangle
rel_x2                   0.75       // gives relative coordinates of rectangle
rel_y1                   0.25       // gives relative coordinates of rectangle
rel_y2                   0.75       // gives relative coordinates of rectangle
dump_rho                 1
…
```

Once all of these variables are entered, the program is ready to be run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation. The output will be found in the out directory in .bmp form and should look similar to the figure below. You will notice that the two components reach equilibrium when the first component forms a circle; this is the state with the lowest interfacial length and thus lowest energy.
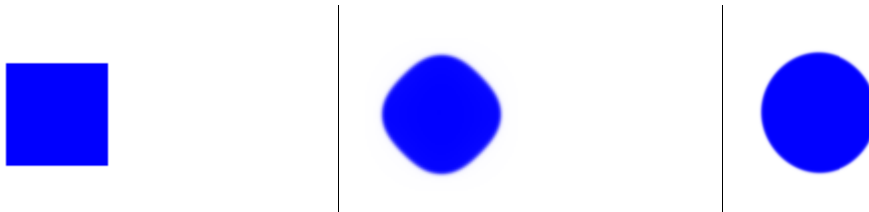

Figure 4: Example of a single solution multi phase separation. Frames from left to right: 0, 400, and 2300 time steps in a 100x100 lu area and depict the density of component one (i.e. [0]).

2.4.4 MULTI-COMPONENT, MULTIPHASE PHASE SEPARATION

We start with an emulsion of two immiscible components and watch as they separate to their lowest energy state.

In Cygwin make sure that you are in the correct folder. Now enter the src folder, rename the file flags_mcmp.h to flags.h (this will already be done if you did the last example), and recompile the program (type 'make' into Cygwin). Next enter the 'in' folder, a new .bmp file must be created here. We are again using a blank area for fully periodic boundaries. Let's use 100x100 lu. Be sure to change the attributes of the .bmp to the correct size (you can use the .bmp that was made for the last example). Also in the in folder open the params.in file.

Params.in

LX	100
LY	100
…	
NumFrames	100
FrameRate	100
tau[0]	1.0
…	
tau[1]	1.0
…	
rho_A	1.0
rho_B	0.0
…	
rho_in	1.0
rho_out	1.0
…	
G	2.0
…	
slice_x	10
slice_y	10
…	
plot_scale_dynamic	1
…	
initial_condition	7
…	
cut	0.1
…	
dump_rho	1
…	

Once all of these variables are entered, the program is ready to be run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation, the output will be found in the 'out' directory in .bmp form and should look similar to the figure below. You will notice that most of the drops are coalescing, however there are a few that seem to disappear; unlike the single component solution these drops are not accounted for by evaporation, but they have diffused into the solute.
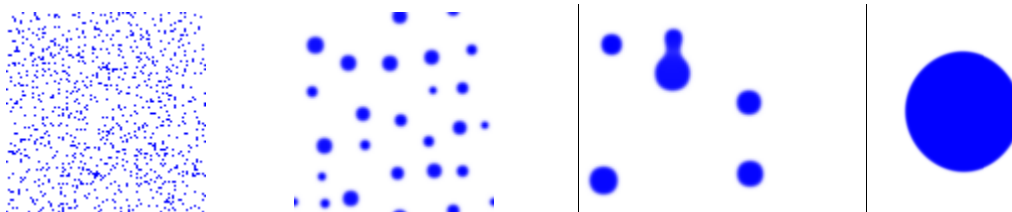

Figure 5: Example of a single solution multi phase separation. Frames from left to right: 0, 2600, 13000, and 23700 time steps in a 200x200 lu area depict the density of component one (i.e., [0]).

2.5   MULTI-COMPONENT, MULTIPHASE CONTACT ANGLE

We begin with a square and observe as it attaches itself to the wall at a 90° angle.

In Cygwin make sure that you are in the correct folder. Now enter the src folder, rename the file flags_mcmp.h to flags.h (this will be done if you did either of the last two examples), and recompile the program (type 'make' into Cygwin). Next enter the 'in' folder, a new .bmp file must be created here. We are again using a blank area. Let's use 100x100 lu. Be sure to change the attributes of the .bmp to the correct size (you can use the .bmp that was made for the last example). Also in the 'in' folder open the params.in file.

Params.in

LX	100
LY	100
…	
NumFrames	100
FrameRate	100
tau[0]	1.0
…	
rho_A	1.0
…	
…	
rho_in	1.0
rho_out	1.0
…	
G	2.0
Gads[0]	-1.0
Gads[1]	1.0
…	
slice_x	50
slice_y	50
…	
plot_scale_dynamic	1
…	
initial_condition	4
x0	25.0
y0	25.0
r0	10.0
…	
dump_rho	1
…	

Once all of these variables are entered, the program is ready to be run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation, the output will be found in the out directory in .bmp form and should look similar to the figure below.
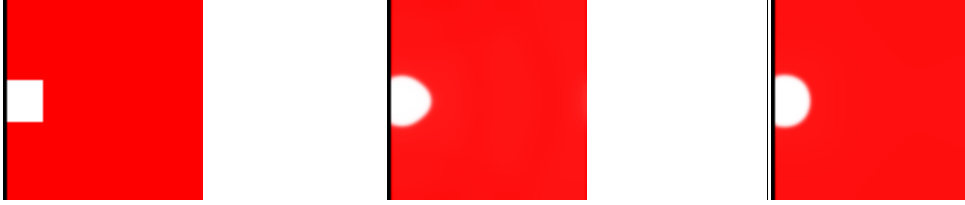
Figure 6: Example of a single solution multi phase surface contact angle of 90°. Frames from left to right: 0, 1, 34 in a 100x100 lu area and depict component one (ie. [1]).

LBM FOR MACROSCPOIC POROUS MEDIA

(Shadab Anwar, Evan Variano)

This option is of value for transcending the limits of pore-scale computational capabilities in the modeling of large scale porous media. It uses a solution that treats the porous medium as a macroscopic continuum.

There are three options for this capability of LB2D_Prime and the choice of these is determined by the selection of the flags that appear in the flags.h file and the params.in file. The POROUS_MEDIA 1 flag must be set in flags.h and the program must be recompiled to activate the porous media options. The three options are:

1. When POROUS_MEDIA 0 in flags.h, the ns_flag and the ns value in params.in are meaningless

2. When POROUS_MEDIA 1 and ns_flag 0 then every pixel gets a resistance equal to the ns value in the params.in file

3. When POROUS_MEDIA 1 and ns_flag 1, then every single pixel gets a resistance determined by its grayscale value in a file named nsLXxLY.bmp, and the ns value in params.in is meaningless

The convention we follow here is that the darker color means higher permeability; hence, white is impermeable and black is infinitely permeable (gray scale value 0 to 255 proportional to resistance). The Navier-Stokes equations will be solved on black nodes where the gray value is 0 and Darcy's law will apply on gray nodes in the range 0 < gray value <255 and white nodes (gray value 255) will act as obstacles. We repeat for clarity: white color means impermeable node for macroscopic porous media modeling, which is the exact opposite of regular pore-solid (binary) domain.

To create an input file that utilizes the gray scale porous medium capability of LB2D_Prime, one option is to open your image file in Microsoft Paint. On the menu bar, click on Colors, Edit Colors, Define Custom Colors, choose zero saturation and then drag the black arrow on the gray-scale to your desired level (0-255). Now click "O.K." and your custom gray-scale is selected and you can paint it into the domain. Save the file as ns_LXxLY.bmp, where LX and LY are the numbers of pixels in the X and Y directions respectively. You must also have a blank (entirely white) image file named LXxLY.bmp in the ./in/ folder.

If the modeler's intent is to use an external file for Ns input, it is recommended to ensure that a warning of the following type is not displayed:

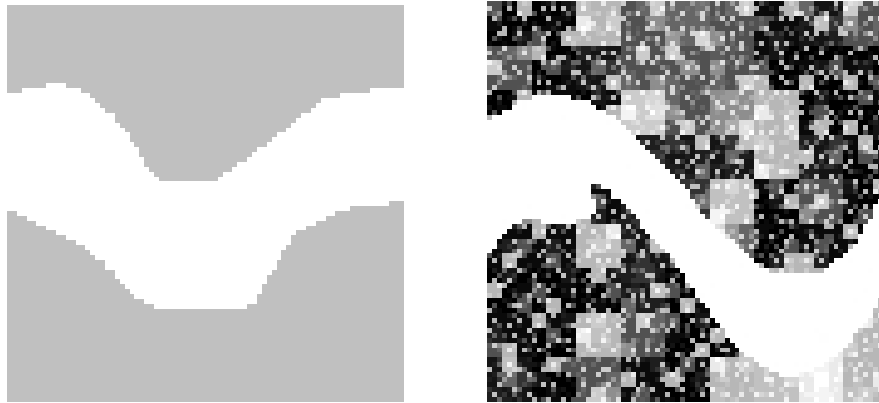./src/latman.c 521 >> WARNING: Can't read "./in/ns616x579.bmp". Using default ns values.

Figure 7: Examples of porous media using ns

2.6   SOLUTE DIFFUSION

In this simulation solute is entered into the flow and allowed to diffuse around a series of solids. The first component is being pulled out at the bottom, while the solute is being introduced at the top of the column and diffusing through to the bottom.

In Cygwin make sure that you are in the correct folder. Now enter the src folder, rename the file flags_mcmp.h to flags.h (this will be done if you did any of the last three examples), and recompile the program (type 'make' into Cygwin). Next enter the 'in' folder, a new .bmp file must be created here, this .bmp is diffucult to produce, **to make it easier there will be a place you can download one already made.** Also in the in folder open the params.in.

Params.in

LX	60
LY	120
…	
NumFrames	50
FrameRate	1000
tau[0]	1.0
…	

tau[1]	0.6	// >0.5
…		
rho_A	1.0	
rho_B	1.0	
rho_sigma	1.0	
rho_sigma_in	1.0	
rho_sigma_out	0.0	
u_sigma	0.0	
u_sigma_in	0.0066	
u_sigma_out	0.0	
sigma_start	0	
sigma_stop	99999999	
sigma_btc_rate	25	
sigma_btc_spot	120	
…		
uy_in	-0.01	
uy_out	-0.01	
…		
slice_x	30	
slice_y	30	
…		
velocity_n_in[0]	1	// velocity BC on north side using u_y_in
…		
Velocity_s_out[0]	1	// velocity BC on south side using uy_out
…		
constcon_n_in	1	
…		
zeroconcgrad_n	0	
zeroconcgrad_s	1	
…		
zeroconcgrad_full	1	
plot_scale_dynamic	1	
…		
initial_condition	2	
…		
dump_rho	1	
…		

Once all of these variables are entered, the program is ready to be run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation, the output will be found in the /out directory in .bmp form and should look similar to the figure below.
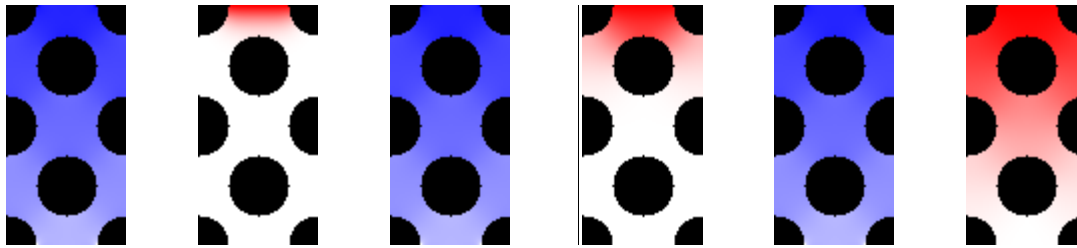
Figure 8: Example of solute diffusion among solids. Frames from left to right: 1, 7, 48 both components (i.e. [0] and sigma).

2.7   BUOYANCY

This simulation is initialized with a region of a dense solute component or low temperature that will begin to sink.

In Cygwin make sure that you are in the correct folder. Now enter the src folder, rename the file flags_mcmp.h to flags.h (this will be done if you did any of the last four examples), and recompile the program (type 'make' into Cygwin). Next enter the 'in' folder, a new .bmp file must be created here, a blank area with a wall along the bottom to prevent the fluid from being pulled out by gravity (for fully periodic boundaries). Let's use 200x200 lu. Be sure to change the attributes of the .bmp to the correct size. Also in the 'in' folder open the params.in file. There are two important aspects of this simulation, the first is that it must run for a long period of time in order to approach equilibrium, and the other is to turn on the buoyancy switch.

Params.in

LX	200
LY	200
…	
NumFrames	1000
FrameRate	1000
tau[0]	0.51
gforce_x[0]	0.0
gforce_y[0]	-0.000006
end_grav[0]	0
tau[1]	0.51
gforce_x[1]	0.0
gforce_y[1]	-0.000006
end_grav[1]	0
buoyancy	1
…	
rho_A	1.0
rho_B	1.0
rho_sigma	1.0
rho_sigma_in	0.0
rho_sigma_out	0.0
…	
rho_in	1.0
rho_out	1.0

```
…
slice_x                  100
slice_y                  100
…
plot_scale_dynamic       1
…
initial_condition        8
…
rel_x1                   0.0
rel_x2                   0.5
rel_y1                   0.0
rel_y2                   1.0
dump_rho                 1
…
```

Once all of these variables are entered, the program is ready to be run. To do this go back to cgywin, ensure that the path is correct and then type './lb2d_prime'. This starts the simulation, the output will be found in the 'out' directory in .bmp form and should look similar to the figure below.
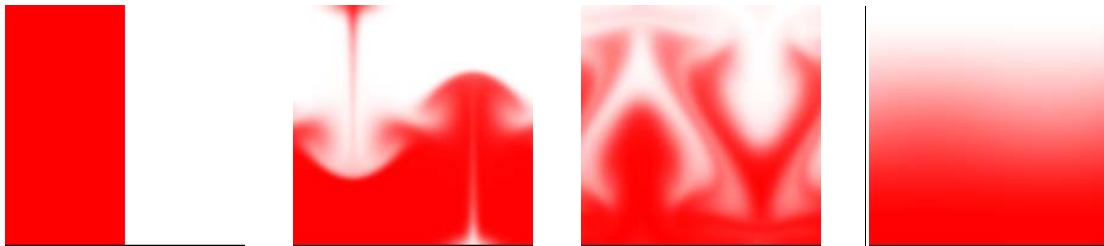

Figure 9: Example of buoyancy. Frames from left to right: 0, 10, 30, 396 in a 200x200 lu area of component 1 (i.e. subs[1]).

3   FLAGS.H FILE FOR COMPILER DIRECTIVES

An important file that has been previously mentioned, but not yet explained is flags.h. To use LB2D Prime efficiently a comprehensive knowledge of this file is required. However for beginners a basic introduction will suffice, as there are many versions of flags.h included in LB2D Prime when the program was installed.

Flags.h is best explained as an include file that contains compile-time switches for various options of LB2D_Prime. Thus, if any changes are made to flags.h, the program must be recompiled (type 'make' in Cygwin) to produce different models. The various options are explained in detail below, many of them can only be turned on (1) or off (0), but a few have other options; these will be explained when applicable. Like params.in, flags.h is edited using any text editor.

The first option is 'verbosity level'. This option is mainly for troubleshooting. It controls how much output is given or 'printed' during program execution. For example if it is set at 0 nothing will be printed, if it is set to 1 only items outside of loops will be displayed, if it is set to 2, items inside the first level of loops will be printed, and so on.

Another method of troubleshooting is the 'say hi' function. If this is on, some routines will display "hi" and "bye" messages. If there is a problem with the simulation this toggle allows the user to see how far the program has run before it failed.

Flags.h needs to be told how many components are involved in a particular simulation. This number can be either 1 or 2, and is set in the 'num_fluid_components' setting. Related to this setting is 'inamuro_sigma_component'. If this option is turned on and there are two components, this switches the second component from a fluid to a solute. By turning on this switch all of the sigma options in params.in now become important.

The Zhang and Chen energy transport method, which outputs thermodynamic consistency and computes body force, a term representing non-local interaction potential speed, is not yet functional.

The porous media simulates a probabilistic bounceback as proposed by Dardis and McCloskey can be turned on by toggling porous_media.

The 'store_ueq' is the true velocity when there is a multiphase and multicomponent system. Therefore, if this is to be used, not only does it need to be on, but also there must be two components.

If your flow area is small compared to the total size of the area, it would be beneficial to have 'do_not_store_solids' option on. This will reduce the storage requirements. However this option is not yet functional in the main distribution.

'Non_local_forces' turns on/off methods for calculating interaction forces, thus it is important when modeling multiphase separation and surface interactions.

'WM' and 'WD' are phase force weighting factors and have been calculated and set at certain values, these should not be changed.

At times it is useful to vary gravity in time, to do this 'manage_body_forces' must be turned on.

When using a solute you must tell LB2D_Prime that you want to use a breakthrough curve. To do this turn on 'store_btc' and ensure the 'inamuro_sigma_component' is on. A value can be assigned to the breakthrough curve in params.in.

Fluid in a model can flow in various directions; to begin with one direction must be stated. This is done in the 'determine_flow_direction' option. This option has three choices. The flow can be vertical (1), horizontal (2) or indeterminate (0). If the flow is indeterminate then the breakthrough curve mentioned above will not be stored.

A model can also be initialized with a flow direction in either the x-direction or the y-direction. This is done using a combination of variables in params.in and flags.h. In flags.h turn on one of 'initialze_with_ux_in' or 'initialze_with_uy_in'. The values for these are set in params.in.

If the simulation can be analyzed by only .bmp and Matlab slice files then is may be useful to turn off write_macro_var_dat_files and write_pdf_dat_files. This will save time and memory.

Another useful tool for debugging is using the write_rho_and_u_to_txt and write_rho_and_u_to_txt. This function should only be used when the simulation requires a very small lattice since it will give density and velocity data at every node of the lattice.

```
// Value used to represent an INACTIVE_NODE .  This is used in the list

// of neighbors ( struct node_struct::nn).  It is also used in the

// map from (i,j) space onto n index space in rho2bmp() and u2bmp().

// Flag: INACTIVE_NODE

#define INACTIVE_NODE -1
```

Real fluid components will have a positive density; puke_negative_densities will stop the simulation if this is not the case. However for long runs it will boost performance if this is turned off.

LB2D_Prime can use different patterns to display solid nodes. To do this turn on the respective toggle, either solid_colour _is_checkerboard or solid_colour _is_black.

```
// Flag: DELAY

#define DELAY    0

// Flag: END_GRAV

#define END_GRAV 2000
```

To track orientation of the simulation a single white pixel can be placed at the origin of the lattice by turning on mark_origin_for_reference.

```
// Flag: PERTURBATIONS

#define PERTURBATIONS 0
```

More information on any of the toggles can be found in LB2D_Prime's documentation folder.

4   PARALLEL LB2D_PRIME

LB2D_Prime is set up to compile and run in an MPI parallel environment. The domain must be broken up into strips perpendicular to the y-axis and the input bmps named as follows from bottom to top: LXxLY_proc0000.bmp, LXxLY_proc0001.bmp, …, LXxLY_proc000n.bmp, where n is one less than the number of processors. In these names, the LY refers to the fraction of the total LY of the domain. For example, a 100x100 domain might be broken up into 2 parts as 100x50_proc0000.bmp and 100x50_proc0001.bmp. The total size is still used in the params.in file, for example, Lx = 100 and Ly = 100.

5   REFERENCE

Sukop, M.C. and D.T. Thorne, Jr., 2006. Lattice Boltzmann Modeling: An introduction for geoscientists and engineers. Springer, Heidelberg, Berlin, New York 172 p.

6   APPENDIX

6.1   POISEUILLE FLOW

The Poiseuille analytical solution provides a fundamental test of single fluid flow simulation. In 2 dimensions it represents the steady flow of a fluid between parallel plates at low Reynolds number. LB2D_Prime automatically computes the analytical solution for the input parameters of channel width, kinematic viscosity, and gravity, when the flow is gravity driven and the domain boundaries perpendicular to the channel walls are periodic (the default). The analytical solution is plotted against the simulation results in the MATLAB 'slice' files only.

To simulate Poisuille flow, go to the 'in' folder and create a new .bmp file. We will try a domain size of 20x40 lu because it will allow for accurate results rather quickly. Once the file is created, open the params.in file that is also located in the 'in' folder. It should look like

Params.in

LX   20

LY   40

characteristic_length   100

NumFrames   10

FrameRate   10000

tau[0]   1.0

gforce_x[0]   0

gforce_y[0]   0.000001

rho_A   1

slice_x   10

slice_y   20

ic_poisseuille   1

bc_poisseuille   0

initial_condition 1

Once this is saved the program is ready to run. Go back to Cygwin and type in './lb2d_prime'. This will start the simulation. Once the simulation is complete you can check to see if the Poiseuille analytical solution graph is correct by using MATLAB. Open MATLAB and change the directory 'cd' to the out folder of the file used in the Cygwin run.
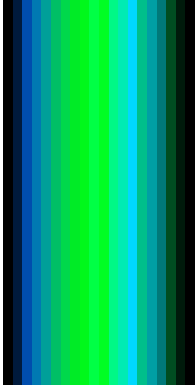
Once this is ready type 'slice_y20x40_frame0010(1)' into MATLAB and press enter. MATLAB Figure window 14 (reproduced here) indicates the Poiseuille analytical solution, which should be lined up.

LB results overlaying analytical Poiseuille profile