# Computational Fluid Dynamics: FHP-I & FHP-III

Mark Alexander Kaltenborn

May 2, 2015

## Introduction to FHP

The FHP model is a type of lattice gas automata(LGA), or lattice gas cellular automata, which is used to simulate fluid flow. These LGAs were early methods that were used to derive the macroscopic Navier-Stokes equations. The interest in these simple models have decreased over the years with development and increasing interest in other models, such as the lattice Boltzmann(LB).

These LGAs consist of a lattice, where, at each site or cell of the lattice, a number of different states can exist. In the LGA, these cells are occupied by particles with a particular velocity. The evolution of the simulation is done in discrete time steps where the lattice cells are evolved by a certain set of rules. At each time step, the state of each site of the lattice is determined and evolved based on the rules in place, before repeating the analysis of the lattice for the subsequent time steps. Each cell of the lattice is a boolean. In other words, there either exists or does not exist a particle in each distinct directional vector.

For each time step, the state of each cell is determined and then two processes are executed: collision and propagation.

For the collision step, if a cell is composed of a collision states, that is, there are multiple particles that occupy a cell and interact with each other, then the cell is evolved to the new post-collision state. These collision rules are required to maintain mass conservation, and conserve the total momentum.

In the propagation step, each particle will be transferred to the neighboring site determined by the velocity of that particle. This process is executed after the collision function. So a particle moves along the direction of its velocity and maintains its velocity in the new cell.

In the 1970s, Hardy, Pomeau and de Pazzis introduced their first attempt at a lattice model, named the HPP model after the authors. The HPP model is a two-dimensional model of fluids. In this model, a square lattice is used to simulate the the particle interactions. Particles have four possible possible velocities and can move to any of the four sites whose cells share a common edge; however, the particles cannot move diagonally in the lattice.

When two particles collide, in a head on collision, for example a particle moving left and a second moving right, the post-collision state is a particle moving up and the second moving down. The HPP model was far too simple to accurately model fluids.

To try and account for the limitations, a hexagonal grid was introduced in the 1980s, in a paper by Uriel Frisch, Brosl Hasslacher and Yves Pomeau. This model was named the FHP model after those who proposed it. This model introduces more velocities, due to the 6 lines connecting each cell in the hexagonal grid. Depending on the variation of the model, there also exists a possible seventh state for the particle, "at rest". The models that account for the resting state are FHP-II and FHP-III. The resting particles do not propagate to neighboring sites, but they are capable of colliding with other particles. The FHP-III model allows all possible collisions, that is 127 separate possible states, that conserve density and momentum. The added collisions in FHP-II and FHP-III allow for simulations of less viscous fluids. The collision rules in the FHP model are not predetermined, some collision states produce two possible post-collision states, and when this occurs, one of the states is picked at random.

## Implementation

C++ was chosen as the language for the purpose of implementing the FHP model. For the rest of this report, a basic level of understanding of the C++ language will be assumed. Included are six files. Three of the files are used for implementing an FHP-I simulation, while the other three are used for FHP-III. (main.cpp, FHP.h, FHP.cpp and main3.cpp, FHP3.h, FHP3.cpp)

The cell states were represented in bit form. This way each each occupied or empty state for each cell would be represented by either a 0 or a 1.

```
M    0x1 // middle (rest)
TL   0x2 // top left
TR   0x4 // top right
R    0x8 // right
BR   0x10 // bottom right
BL   0x20 // bottom left
L    0x40 // left
DSB  0x80 // disabled cell (boundary, obstacle)
```

The implementation follows very simply from there. The program consists of eleven important functions: Cell(), Lattice(), PartCount(), FirstRandState(), RandState(), Propagate(), Collide(), CalcVelocity(), Step(), PrintVelocity(), GetState().

Cell():

The cell must first be defined, as well as, all operations that are performed on the cells.

Lattice():

Once all properties of the cell have been defined, a hexagonal lattice of cells is then created. The boundary conditions of the lattice are set. The first state of the cell is also called in this function.

PartCount():

This function is responsible for counting the number of particles of all states that reside in the given boundaries.

FirstRandState():

This function is used to create a first state of the lattice. This is set separately from the function RandState() which can be called after each time step.

RandState():

In order to maintain a specific pressure difference in my pipe, this function checks that the $\Delta P$ remains constant during each time step. It is also useful to separate the first state from future states. This can allow for a perturbation of the original state after each time step.

Propagate():

The propagation function is quite straight forward. It contains the rules for transferring each particle to a new cell after each step. This is also where collision rules with boundaries/obstacles are set.

Collide():

The collision function is the most important function in determining which particular FHP model is being implemented.

CalcVelocity():

In order to acquire a velocity profile, a section of the pipe is observed and the magnitude of velocity is calculated. The data collected from here is used later to draw plots.

Step():

This function calls each time step of the simulation. Here it is important to call the functions in their proper order (Collide, Propagate, CalcVelocity, and then RandState).

PrintVelocity():

It is here that the data from the simulation is collected and printed into separate datafiles which can then be plotted with your preferred application.

GetState():

This function gets the value of the state of a cell. This is useful for the graphical interface that was chosen to use.

gui():

This is the function for visualizing the fluid in the terminal window.

In the main(3).cpp, a simple option for a progress bar can be turned on and off. The progress bar is quite useful for long simulations.

## Velocity Profiles for Laminar Flow

Multiple attempts were made of each program to assess properties of the fluid. A common application of laminar flow is in the smooth flow of a viscous liquid through a tube or pipe, where we treat our 2D grid as a flat section of a cylindrical tube. In this case, the velocity of flow varies from zero at the walls to a maximum along the cross-sectional centre of the vessel. The flow profile of laminar flow in a pipe was calculated with the FHP models mentioned above.

A little background information for the simulations.

Consider a cylinder of fluid, length $L$, radius $r$, flowing steadily in the centre of a pipe. In equilibrium, the shearing forces on the cylinder equal the pressure forces $P$.

$$\tau 2\pi r L = \Delta P A = \Delta P \pi r^2$$
$$\tau = \frac{\Delta P}{L}\frac{r}{2}$$

By Newton's law of viscosity, we have that $\tau = \mu \frac{dv}{dx}$, where $x$ is the distance from the wall of the pipe. Since the pipe has a radius $r$, we make a substitution, giving

$$\tau = -\mu \frac{dv}{dr}$$

Combining this with the above equation, we get

$$\frac{\Delta P}{L}\frac{r}{2} = -\mu \frac{dv}{dr}$$
$$\frac{dv}{dr} = -\frac{\Delta P}{L}\frac{r}{2\mu}$$

Integrating this function we get an expression for velocity, $v$ and then evaluating gives a value of velocity from the center by a distance $r$

$$v = -\frac{\Delta P}{L}\frac{1}{2\mu}\int r dr$$
$$v_r = -\frac{\Delta P}{L}\frac{r^2}{4\mu} + \text{Const.}$$

At $r = 0$, $v = v_m ax$, at $r = R$ (the pipe wall) $v = 0$, giving

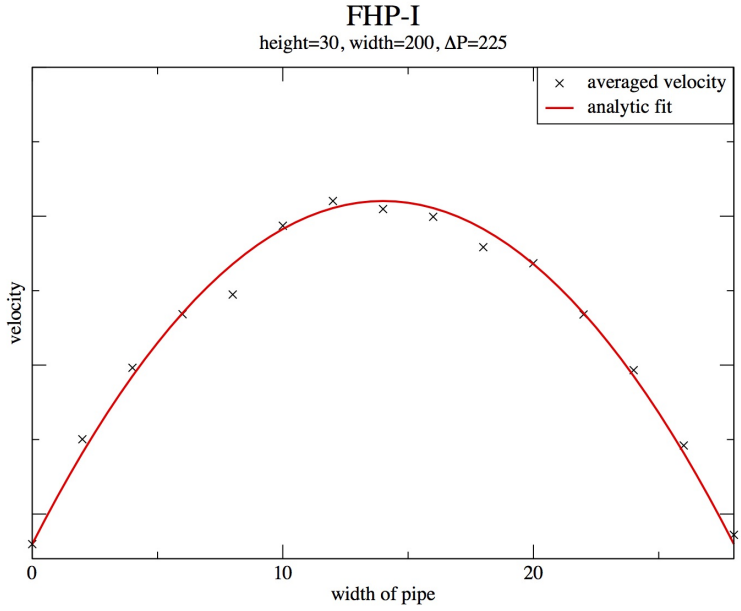
$$\text{Const.} = \frac{\Delta P}{L}\frac{R^2}{4\mu}$$

So, we finally get an expression for velocity at a point r from the pipe centre when the flow is laminar

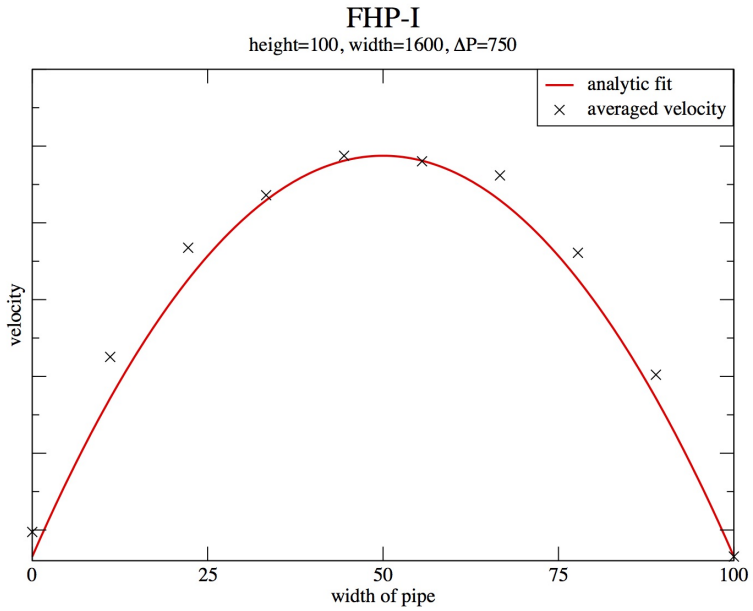
$$v_r = \frac{\Delta P}{L}\frac{1}{4\mu}(R^2 - r^2)$$

Now we shall compare results taken from simulations of the FHP models and compare to an analytic fit of the form of the equation derived quickly above.

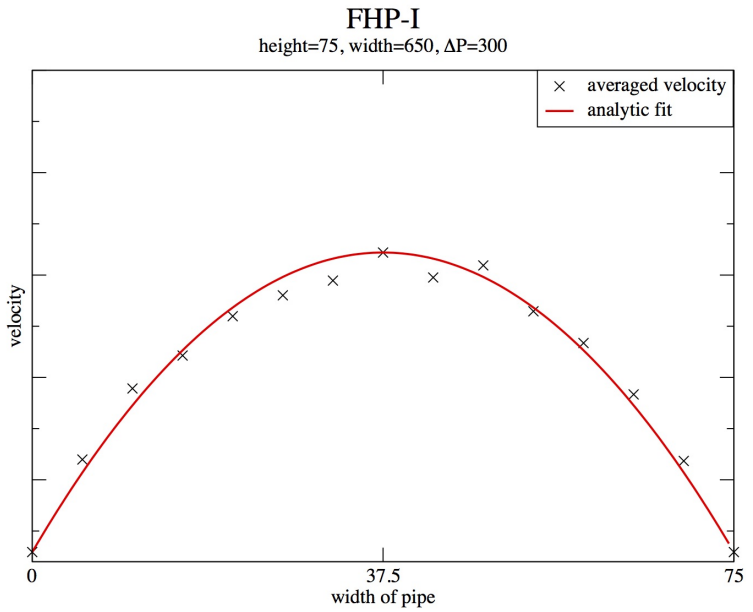
## Results



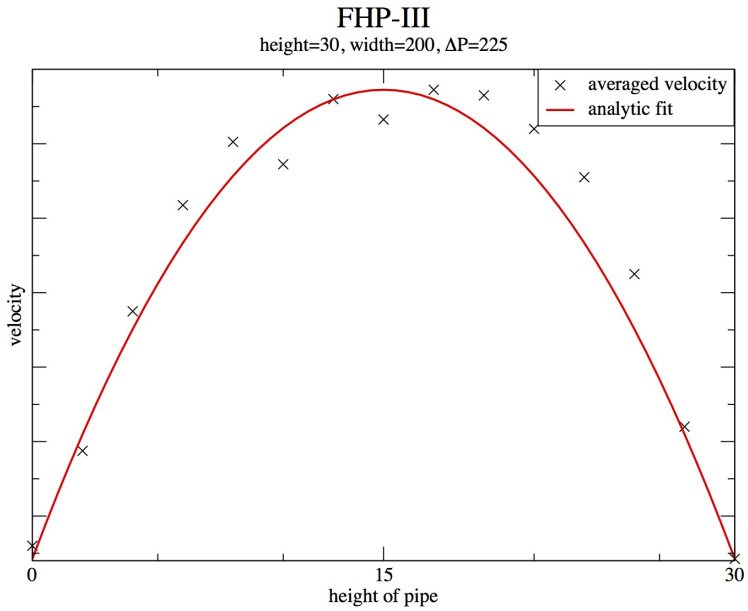

(Fig. 1).


FHP-I
height=100, width=1600, ΔP=750
analytic fit
averaged velocity
velocity
0
25
50
75
100
width of pipe

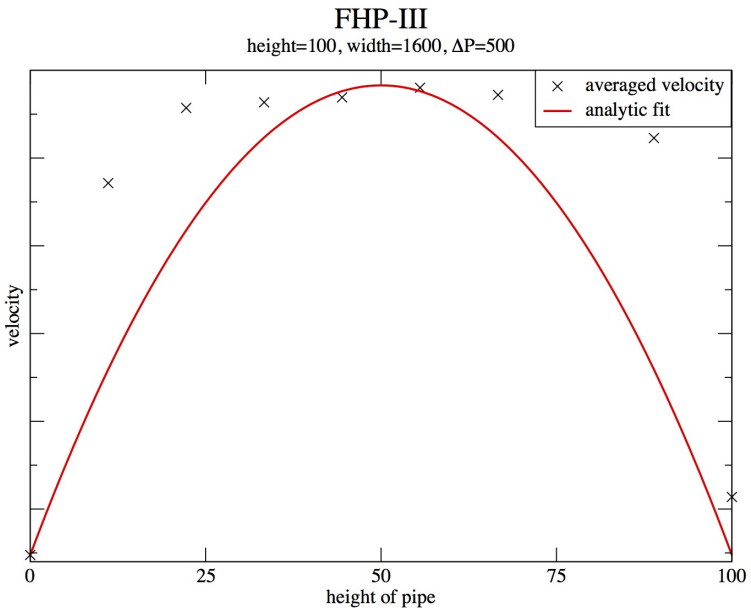
(Fig. 2).


FHP-I
height=75, width=650, ΔP=300
averaged velocity
analytic fit
velocity
0
37.5
75
width of pipe

(Fig. 3).


FHP-III
height=30, width=200, ΔP=225
averaged velocity
analytic fit
velocity
0
15
30
height of pipe

(Fig. 4).


FHP-III
height=100, width=1600, ΔP=500
averaged velocity
analytic fit
velocity
0
25
50
75
100
height of pipe

(Fig. 5).

## Conclusion

From the plots above it is quite clear that the FHP-I model does quite a decent job fitting the velocity profile expected (Fig. 1,2,3). However, from the the plots for the FHP-III model, we see that the for narrow pipes the model holds, but as the pipe widens there an issue arrises from the profiles(Fig. 4,5).