**2-D Fluid Dynamics Simulator Application**

# Software Architecture Document

# *Programming Guide*

**Tech Geeks**

# Presented by:

# Nagasindhu Kannekanti

# Ayman Almusalam

# Michael Chase Bonifant

# Sai Krishna Kalakonda

# Srikanth Reddy Bitla

# Prepared for:

# Principles of Software Engineering

# FALL 2015

# 1. Introduction:

This document is to present the programmer guide for 2-D Fluid Dynamics Simulator system. It will explain all programming code and features for 2-D Fluid Dynamics Simulator system. In this application we used Lattice Boltzmann Methods (LBM) which is a class of computational fluid dynamics methods used in fluid simulation from which the Navier-Stokes equations can be derived. Chamber: when used in this document chamber refers to the simulated container that the fluid dynamics are being simulated inside of. System: when used in this document refers to the deliverable end product, the 2D Fluid Dynamics Simulator.

# 2. General Description

## 2.1 Product Perspective

This system will serve as a self-contained desktop application for simulating 2-D fluid dynamics. It should be a simple, intuitive tool that allows users to configure a chamber filling or filled with fluid, vary environmental factors (e.g. obstructions in the chamber, temperature, viscosity), then apply a force vector to disturb the fluid and show a simulation of that force’s effect on the fluid.

## 2.2 Product Functions

The system allows the user to create a simulation chamber for testing what happens when a force vector is applied to a fluid filling that chamber. As such the system allows users to select a fluid and a force vector. Additionally they can set environmental variables such as the temperature among others (see section 3.2.2 for a full listing of configurable variables). The user can also create within the chamber any number of physical obstructions the fluid can potentially rebound against and flow around.

The user can then set monitors at various points in the chamber. Once the chamber is fully configured the user can launch the simulation and run it until they choose to terminate the simulation. The simulation runs showing a visual display of the created chamber and also provides a textual print out at the state of the simulation at all break points.

From the textual print out the will be able to reload the simulation and replay the simulation.

## 2.3 User Characteristics

1. Users for this system include students in high school and beyond learning the basics of fluid dynamics and their teachers.

## 2.4 General Constraints

The main constraints are the system must be a desktop application, and it must run on either Linux or Windows. Beyond that the simulation must be based on the LBM.

**3. Coding explanation:**

With JAVA here we are going to explain each main section in this application. This application is divided into six main sections, which are fluids, GUI, lattice, logger, parsers, and tests.

**3.1 Fluids**

I this section we are going to divided the liquid into two different class or types which users can chose between them. The two class are glycerin and water. Also liguid interface.

**3.1.1 Liquid**

In this section we are going to writer a class that get viscosity, density, omega, temperature and set a temperature from user.

|  |  |  |
| --- | --- | --- |
| public interface Liquid { | |  |
| public float getViscosity(); |
| public float getDensity(); |
| public float getOmega(); |
| public int getTemperature(); |

public void setTemperature(int t);

**3.1.2 Glycerin**

|  |  |  |
| --- | --- | --- |
| TODO add in more values of viscosity at varying | |  |
| \* temperatures |
| public class Glycerin implements Liquid { |
| /\*\* A Map of the kinematic viscosity of waters temperature increases \*/ |
| public static float[] viscosityFromTemp = { -1,//0C |
| -1,//10C  Here a map of kinematic viscosity of water temperature increases.  /\*\* water temperature \*/  private int temperature;  /\*\* water density \*/  private float density;  /\*\* water viscosity (how well it flows) \*/  private float viscosity;  /\*\* time constant related to viscosity by omega= 1/(3\*viscoity+0.5) \*/  private float omega;  after that we need to open a class that synchronized a temperature. |

**3.1.3 Water**

TODO add in more values of viscosity at varying temperatures

public class Water implements Liquid {

/\*\* A Map of the kinematic viscosity of waters temperature increases \*/

public static float[] viscosityFromTemp // Here a map of kinematic viscosity of water temperature increases.

= { 1.7918065f, 1.3062651526112f,1.0034053420487f,

0.8008271785568f, 0.65809618509849f, 0.553468039063689f,

0.47436738184881f, 0.41307954234706f, 0.36463671901831f, 0.32570900878262f,0.29688726487272f};

Same as Glycerin private class we have temperature, density, viscosity and omega.

Here in this public class we synchronized a temperature from user inputs.

public synchronized void setTemperature(int t) {

this.temperature = t; // t is the temperature of the water.

this.density = 1;

this.viscosity = viscosityFromTemp[t / 10];

this.omega = 1f / (3f \* this.viscosity + 0.5f);

System.out.println(this.viscosity);

}

**3.2 Graphic User Interface (GUI)**

In GUI section we are going to create ten classes which are boundary, color stats, configurations, drawing toll, edit lattice, feature flags, LBM GUI, main and message bar.

**3.2.1 Boundary**

**3.2.1.1 Class Boundary**

#public class **Boundary**

Boundaries are rectangular shapes, ellipses and actual rectangles drawn on the screen when the user is setting up their chamber configuration. They first draw a rectangle for the chamber and then add ellipses and rectangles inside it as obstructions.

**3.2.1.2 Constructors**

#[**Boundary**](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/Boundary.html#Boundary-gui.Feature-float-float-float-float-)([**Feature**](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/Feature.html) rect,float x,float y, float width, float height)

**Parameters:**

rect - the boundary is a rectangle or else an ellipse, and either a wall, sink or source

x - initial x coordinate (left-most x coordinate)

y - initial y coordinate (top-most y coordinate)

width - width of this Boundary

height - height of this Boundary

**3.2.1.5 Get Area**

public float getArea()

**Returns:**

the area this Boundary covers in pixels^2.

3.2.1.4 Set Feature

public void setFeature([Feature](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/Feature.html) f)

Set the given feature and shift from Rectangle to Ellipse or vice versa

**Parameters:**

f - the feature to shift the Boundary to

**3.2.2 Color Stats**

public ColorStats()

To get the color.

**3.2.3 Configurations**

**3.2.3.1 Class Configurations**

#public class **Configurations**

Configurations provides static methods for obtaining and saving user preferences as well as modification time and image groupings at closing of the ILB (Image List Browser).

**3.2.3.2 Method**

publicstatic void getDisplayProperties([LBMGui](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/LBMGui.html) gui, java.lang.String fileName)

Loads the initial location and size of the ImageListBrowser and sets them.

**Parameters:**

iLB - the browser whose fields will be set

fileName - name of the configuration file

**3.2.4 Drawing Tool**

**3.2.4.1 Class Drawing Tool**

public class **DrawingTool**

A Mouse and Key Listener, allows users to draw ellipses and rectangles on a canvar, and label them as either sources, sinks, or walls.

**3.2.4.2 Constructor**

public DrawingTool([EditLattice](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/EditLattice.html) eL)

**Parameters:**

eL - this DrawingTools EditLattice

**3.2.5 Edit Lattice**

**3.2.5.1 Class Edit Lattice**

public class **EditLattice**

Opens a specified image to its fullest size possible according to screen dimensions in a separate JFrame from the ILB.

EditImage windows cannot be resized.

EditImage uses the [DrawingTool](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/DrawingTool.html) as a MouseListener and KeyListener for editing image Annotations.

**3.2.6 Feature Flags**

**3.2.6.1 Class Feature Flags**

public class **FeatureFlags**

Flags indicating metadata about specific features.

**3.2.7 Lattice Panel**

**3.2.7.1 Class Lattice Panel**

#public class **LatticePanel**

Implements [RefreshListener](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/gui/RefreshListener.html) listener that tells an object how to refresh/update.

**3.2.7.2 Constructor**

public LatticePanel([Lattice](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/lattice/Lattice.html) l,

int dens,

float factor,

[Lattice.ColorStats](file:///Users/mac/Downloads/Fluid2D-master/LatticeBoltzmann/doc/lattice/Lattice.ColorStats.html) c)

**Parameters:**

l - Lattice thie panel draws

dens - number of pixels squared per cell

factor - factor to apply to the getHSV to intensify differences in macro/microscopic values of a cell

c - which macro/microscopic value to view.

**3.2.8 LBM GUI**

**3.2.8.1 Class LBM GUI**

public class **LBMGui**

Opens the application and defines the main window.

**3.2.9 Main**

**3.2.9.1 Class Main**

public class **Main**

Launch and run the ILB.

**3.2.9.2 Method**

public static void main(java.lang.String[] args)

Sets the UI look & feel to match current OS as best as possible. Then launches the application and loads the last session.

**Parameters:**

args – Ignored

**3.2.10 Message Bar**

**3.2.10.1 Class Message Bar**

public class **MessageBar**

The Message bar is a simple panel with a progress bar for showing the loading status of the current background thread, and a JLabel for relevant errors that users might like to know, like if Face Match didn't load.

**3.2.10.2 Constructor**

public MessageBar()

Init the Message bar and set the progress and message to visible.

**3.3 Lattice**

A Lattice is a 3 Dimensional Array defining a collection of Cells through which fluid propagates as an approximation of Fluid Dynamics. Imagine each cell has a little packet of fluid in it, the lattice tracks how much fluid moves into each cell and at what rate.

public class Lattice {

float four9ths = 4.0f / 9.0f;

float one9th = 1.0f / 9.0f;

float one36th = 1.0f / 36.0f;

**3.4 Logger**

**3.4.1 Lattice Logger**

|  |  |  |
| --- | --- | --- |
| /\*\* log information about a given lattice \*/ | |  |
| public class LatticeLogger { |
| \* @param l |
| \* the lattice to log |
| \* @param obj |
| \* other things to look at, change as need be. (ie: flow meters) |
| public static void log(Lattice l, Object... obj) { |
| } |

**3.4.2 Lattice Logger Parser**

|  |  |  |
| --- | --- | --- |
| ackage logger; | |  |
| /\*\* Parses Lattice Logs into Lattices \*/ |
| public class LatticeLoggerParser { |
| } |

**3.5 Parsers**

**3.5.1 Fluid Parser**

|  |  |  |
| --- | --- | --- |
| package parsers; | |  |
| \* A collection of Parsers for reading XML files that give information about a |
| \* fluid (mainly, its name and viscosity at given temperatures). |
| public class FluidParser { |

3.6 Tests

In this section we are going to test the lattice structures by adding shapes and ensuring they are added.

ArrayList<Boundary> arr = new ArrayList<>();

JFrame f = new JFrame(); // arr.add(b4)

LatticePanel p = new LatticePanel(l, 50, 1024, Lattice.ColorStats.speed);

f.add(p);

f.pack();

f.setVisible(true);

|  |  |  |
| --- | --- | --- |
| // Timer t = new Timer(1000, new ActionListener() { | |  |
| // public void actionPerformed(ActionEvent evt) { |
| // p.repaint(); |
| // } |
| // }); |

// t.start();

l.tm.setDelay(250);

l.tm.setInitialDelay(0);

l.tm.start();

// wait for input to stop

try {

System.in.read();

} catch (IOException e) {

e.printStackTrace();

)