NUMERICAL SIMULATION OF THE CASCADE AERATOR SYSTEM FOR THE REMOVAL OF IRON AND MANGANESE

RHAHIMI BINTI JAMIL

UNIVERSITI SAINS MALAYSIA

2019

NUMERICAL SIMULATION OF THE CASCADE AERATOR SYSTEM FOR THE REMOVAL OF IRON AND MANGANESE

by

RHAHIMI BINTI JAMIL

Thesis submitted in fulfillment of the

requirements for the degree of

Doctor of Philosophy

July 2019

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor, Assoc. Prof. Dr. Mohd Remy Rozainy b. Mohd Arif Zainol, and my co-supervisors Prof. Ir. Dr. Mohd Nordin Adlan and Dr Mohamad Aizat b. Abas (School of Mechanical Engineering) for their continuous support of my PhD study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in my time of research and writing this thesis. I cannot imagine having better advisors and mentors for my PhD study.

I am thankful to En. Nizam, who assisted me during the model setup in the lab. I am also grateful to En. Mohd, En. Taib, En. Zaini, Pn. Samsiah and other laboratory staff who helped me along the way. A special thanks to En. Zul from Rumah Nur Kasih, who allowed me to collect the samples at their location.

I am grateful to my husband, Mohd Hilman bin Abdullah, my siblings, and mother, Puteh bt. Abdul Razak, who have always provided me moral and emotional support throughout my life thus far. I am also grateful to my other family members and friends who have supported me along the way.

A very special thanks goes out to all down at the Research Fund, and also the scholarship, for helping me by providing the funding for the research, especially Kementerian Pengajian Tinggi Malaysia.

Last but not least, my sincere thanks also goes to the LRGS team (LRGS grant No. 203/PKT/6726001 – Riverbank Filtration for Drinking Water Source Abstraction), who provided me the opportunity to join their team as an intern, and gave me access to the laboratory and research facilities. Without their precious support, it would not be possible to conduct this research. Thanks for all your encouragement!

TABLE OF CONTENTS

CHAPTER FOUR: RESULT AND DISCUSSION

CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CFD	Computational Fluid Dynamics
LBM	Lattice Boltzmann Method
GWB	Geochemist's Workbench
DPM	Dispersed Phase Method
3D	Three Dimensional
Fe	Iron
$Fe(OH)_3$	Iron (III) hydroxide or ferric acid
ICP	Inductively Coupled Plasma
ICP-OES	Inductively Coupled Plasma-Optical Emission Spectrometer
DO	Dissolved Oxygen
COD	Chemical Oxygen Demand
BOD	Biochemical Oxygen Demand
Mn	Manganese
MnO_2	Manganese (IV) dioxide or manganic oxide
PIV	Particle Image Velocimetry
VOC	Volatile Organic Compound
WHO	World Health Organisation

LIST OF SYMBOLS

Latin Symbols

c_n	Velocity set
E_{20}	Aeration efficiency
ΔE	Energy
f^{eq}	Local equilibrium distribution function
(r,c,t)	Function of distance, velocity, and time
(…)	Distribution function
f_n	Velocity component in distribution function
g	Gravity
G	Interaction strength
ΔH	Velocity head
I(x)	Image intensity field
l_x	Length in x-axis
N	Resolution
P	Energy dissipation rate
ΔP	Pressure
Q	Flow rate of water
r	Radius of bubble
R	Universal gas constant
R(s)	Cross-correlation of two frames
s	Separation vector
T	Temperature
Δt	Time difference
u	Velocity
$V_0\ (Xi)$	The transfer function for the light energy of an individual particle of an image inside the interrogation volume

ω	Collision frequency
$\boldsymbol{\varepsilon}$	Energy dissipation rate per unit width
Δx	Common particle displacement vector

Greek Symbols

τ	Relaxation factor
τ(x - xi)	Point spread function of the imaging lens
σ	Surface tension
ρ	Density
δx	Particle image displacement
Ω	Collision
ψ	Interaction potential

SIMULASI BERANGKA SISTEM LATA PENGUDARAAN BAGI PENYINGKIRAN BESI DAN MANGAN

ABSTRAK

Dalam rawatan air bawah tanah, banyak teknik telah disiasat, termasuk sistem Lata pengudaraan, yang merupakan teknik untuk menghilangkan logam berat seperti besi dan mangan. Lata pengudaraan juga digunakan sebagai kaedah yang berkesan, kos rendah untuk merawat air bawah tanah. Dalam kajian ini, Kaedah Lattice Boltzmann (LBM) telah digunakan untuk menyiasat proses pengudaraan dalam model lata pengudaraan yang baru direka bentuk. Untuk lata pengudaraan baru ini, dimensi yang berbeza telah digunakan untuk menentukan reka bentuk terbaik yang boleh mengurangkan kepekatan besi dan mangan. Dua set simulasi LBM, dan dua set eksperimen Velocimetry Gambar Zarah (PIV) telah dijalankan, dan halaju aliran dikira. Berdasarkan penemuan, ditunjukkan bahawa simulasi LBM, dan data PIV berada dalam persetujuan yang baik antara satu sama lain dari segi pengedaran halaju. Di samping itu, ia juga mendapati bahawa halaju air mempunyai pengaruh yang signifikan terhadap kecekapan pengudaraan. Peronggaan tersebut merosakkan struktur limpahan lonjakan, dan proses pengoksidaan mengurangkan besi dan mangan di dalam air dengan meningkatkan oksigen terlarut. Dalam kajian ini, dua model fizikal lata pengudaraan digunakan, Model A dan Model B, dengan kadar aliran 1.78 l/j, 2.0 l / j, dan 2.20 l / j. Kepekatan oksigen terlarut meningkat dari 0.8 kepada 1.4 mg / L untuk Model A, dan dari 0.7 kepada 1.2 mg / L untuk Model B. Pengeluaran kepekatan besi dan mangan adalah 11.3 mg / L sehingga 16.3 mg / L (2%) , dan 0.31 mg / L sehingga 0.50 mg / L (21%). Untuk perbandingan yang lebih komprehensif, Model C adalah dicipta menggunakan simulasi LBM. Saiz panjang langkah dalam Model C adalah lebih panjang daripada Model A dan Model B. Selebihnya dimensi dalam Model C

adalah serupa dengan dua model yang lain. Secara keseluruhan, panjang Model C ialah 400 mm. Peningkatan yang ketara dapat dilihat dari peratusan penyingkiran yang dicatatkan oleh Model C, iaitu 50% hingga 52% lebih tinggi daripada yang dicatatkan dalam Model A, dan 55% kepada 63% lebih tinggi daripada Model B. Model-model ini telah direka menggunakan Dynamics Fluid Computational (CFD) untuk simulasi berangka. Analisis CFD membenarkan ramalan kehadiran besi dan mangan dalam model lata pengudaraan. Simulasi zarah besi dan mangan dilakukan menggunakan Kaedah Tahap Dispersed (DPM). Hasil yang diperolehi dengan kehadiran besi (10%) dan mangan (5%) dikira dari simulasi. Keputusan kehadiran besi dan mangan hampir sama dengan kerja percubaan sebenar. Oleh itu, CFD digunakan dengan jayanya sebagai alat untuk reka bentuk, dan ramalan kehadiran zarah dalam lata pengudaraan. Di samping itu, dengan menggunakan perisian Avogadro, interaksi antara zarah-zarah yang kelihatan dan pembacaan pengoptimuman geometri diperolehi berdasarkan keputusan tindak balas antara air, besi, dan mangan. Ini menunjukkan bahawa penggunaan perisian Avogadro dapat membantu dalam memerhatikan tindak balas antara zarah air, besi, dan mangan. Keadaan ini dapat dilihat dengan penambahan zarah-zarah yang menunjukkan peningkatan penghapusan besi dan mangan di dalam air bawah tanah.

NUMERICAL SIMULATION OF THE CASCADE AERATOR SYSTEM FOR THE REMOVAL OF IRON AND MANGANESE

ABSTRACT

In groundwater treatment, many techniques have been investigated, including the cascade aerator system, which is a technique to eliminate heavy metals such as iron and manganese. The cascade aerator is also used as an effective, low-cost method to treat groundwater. In this study, the Lattice Boltzmann Method (LBM) was used to investigate the aeration process in a newly-designed cascade aerator model. For this new cascade aerator, different dimensions were used to determine the best design that could reduce the concentration of iron and manganese. Two sets of LBM simulations, and two sets of Particle Image Velocimetry (PIV) experiments were carried out, and the velocity of the flow were calculated. Based on the findings, it was shown that the LBM simulations, and PIV data were in good agreement with each other in terms of velocity distribution. In addition, it was also found that water velocity had a significant influence on aeration efficiency. Cavitation damaged the overflow structure of the surge, and the oxidation process reduced the iron and manganese in the water by increasing the dissolved oxygen. In this study, two physical models of a cascade aerator were used, Model A and Model B, with flow rates of 1.78 l/h, 2.0 l/h, and 2.20 l/h. The dissolved oxygen concentration was increased from 0.8 to 1.4 mg/L for Model A, and from 0.7 to 1.2 mg/L for Model B. The removal of iron and manganese was increased from 11.3 mg/L up to 16.3 mg/l (2%), and 0.31 mg/L up to 0.50 mg/l (21%) respectively. For a more comprehensive comparison, Model C was explored using LBM simulations. The length of the steps in Model C was longer than Model A and Model B. The rest of the dimensions in Model C were similar to the other two models. Overall, the length of the Model C was 400 mm. A significant increase could be

observed from the removal percentages recorded by Model C, which were 50% to 52% higher than those recorded in Model A, and 55% to 63% higher than Model B. These models were designed using Computational Fluid Dynamics (CFD) for numerical simulation. The CFD analysis allowed for the prediction of the presence of iron and manganese in the cascade aeration model. Simulations of iron and manganese particles were performed using the Dispersed Phase Method (DPM). Results obtained on the presence of iron (10%) and manganese (5%) were computed from the simulation. The results on the presence of iron and manganese was almost identical to the actual experimental work. Therefore, CFD was used successfully as a tool for design, and prediction of the presence of particles in the cascade aerator. In addition, with the use of the Avogadro Software, the interactions between the visible particles, and geometric optimisation readings were obtained based on the results on the reactions between water, iron, and manganese. This showed that the use of the Avogadro Software can help in observing the reactions between water, iron, and manganese particles. This can be seen with each addition of particles showing an increase in the removal of iron and manganese in groundwater.

LIST OF APPENDICES

APPENDICES

Appendix A: Concentration Saturation Value of Dissolved Oxygen in Freshwater Exposed to a Saturated Atmosphere

Concentration Saturation Value of Dissolved Oxygen in Freshwater Exposed to a Saturated Atmosphere Containing 20.9% Oxygen under a Pressure of 101.325 kPa

Temperature (°C)	Concentration saturation of dissolved oxygen, C_s (mg/L)	Saturated vapor pressure (kPa)
0	14.62	0.6108
1	14.23	0.6566
2	13.84	0.7055
3	13.48	0.7575
4	13.13	0.8129
5	12.80	0.8719
6	12.48	0.9347
7	12.17	1.0013
8	11.87	1.0722
9	11.59	1.1474
10	11.33	1.2272
11	11.08	1.3119
12	10.83	1.4017
13	10.60	1.4969
14	10.37	1.5977
15	10.15	1.7044
16	9.95	1.8173
17	9.74	1.9367
18	9.54	2.0630
19	9.35	2.1964
20	9.17	2.3373
21	8.99	2.4861
22	8.83	2.6430
23	8.68	2.8086
24	8.53	2.9831
25	8.38	3.1671
26	8.22	3.3608
27	8.07	3.5649
28	7.92	3.7796
29	7.77	4.0055
30	7.63	4.2430
31	7.51	4.4927
32	7.42	4.7551
33	7.28	5.0307
34	7.17	5.3200
35	7.07	5.6236
36	6.96	5.9422
37	6.86	6.2762
38	6.75	6.6264

(*Source*: Calculated by G. C. Whipple and M. C. Whipple from measurements of C. J.J. Fox, *Journal of the American Chemical Society,* vol.33, p.362, 1911 cited in Davis & Cornwell 2013, p.1013)

Appendix B: Aeration Efficiency Data

Variables		Tempe-rature	Aeration Efficiency			
Cascade Heighth, H (m) 0.865 m	Flow rate, Q_w (m^3/hr)	(°C)	Tank 1: Upstream DO, C_u (mg/L)	Tank 4: Upstream DO, C_d (mg/L)	C_d - C_u (mg/L)	Aeration Efficiency, E_{20}
A	1.78/1.11	30.8	1.3	3.4	2.1	0.29
A	1.78/1.11	30.8	1.1	3.4	2.3	0.30
A	1.78/1.11	30.8	1.3	4.6	3.3	0.46
A	1.78/1.11	30.8	1.3	3.9	2.6	0.36
B	2.0/1.33	30.8	1.6	3.6	2.0	0.29
B	2.0/1.33	30.8	1.4	3.6	2.2	0.30
B	2.0/1.33	30.8	1.6	3.6	2.0	0.29
B	2.0/1.33	30.9	1.5	3.6	2.1	0.29
C	2.0/1.55	30.9	1.6	3.6	2.0	0.29
C	2.0/1.55	30.9	1.5	3.6	2.1	0.29
C	2.0/1.55	30.9	1.6	3.6	2.0	0.29
C	2.0/1.55	30.9	1.7	3.7	2.0	0.29
D	1.78/1.11	31.1	1.3	3.5	2.2	0.30
D	1.78/1.11	31.1	0.8	3.3	2.5	0.31
D	1.78/1.11	31.1	0.9	3.3	2.4	0.31
D	1.78/1.11	31.1	1.2	3.3	2.1	0.28
E	2.0/1.33	31.0	1.2	3.5	2.3	0.31
E	2.0/1.33	31.0	1.3	3.5	2.2	0.30
E	2.0/1.33	31.0	1.5	3.7	2.2	0.31
E	2.0/1.33	31.0	1.4	3.7	2.3	0.32
F	2.22/1.55	30.9	1.5	3.7	2.2	0.31
F	2.22/1.55	30.9	1.8	3.9	2.1	0.31
F	2.22/1.55	30.9	1.8	3.7	1.9	0.28
F	2.22/1.55	31.0	1.6	3.7	2.1	0.30

Appendix C: Velocity result of simulation and experiment for Model A and Model B

Velocity result of simulation and experiment for Model A

Point	Velocity, m/s		Location of the Point		
	Simulation	Experiment	X	Y	Z
A	0.242	0.258	94	39.5	72
B	0.248	0.270	77	39.5	68
C	0.323	0.361	63	39.5	64
D	0.378	0.389	64	39.5	63

Velocity result of simulation and experiment for Model B

Point	Velocity, m/s		Location of the Point		
	Simulation	Experiment	X	Y	Z
A	0.293	0.301	93	39.5	72
B	0.318	0.346	77	39.5	68
C	0.456	0.475	65	39.5	64
D	0.495	0.501	66	39.5	63

Appendix D: Simulation Pressure for Model A and Model B

Pressure in simulation results for Model A.

Model	Pressure, Pa	Point of location		
		X	Y	Z
A	1.533	93	39.5	72
B	1.524	77	39.5	68
C	1.458	63	39.5	64
D	1.360	64	39.5	63

Pressure in simulation results for Model B.

Model	Pressure, Pa	Point of location		
		X	Y	Z
A	1.396	94	39.5	72
B	1.152	76	39.5	68
C	1.054	65	39.5	64
D	0.683	65	39.5	63

EPA Guidelines: Water and wastewater sampling

Analyte	Container type	Typical volume (mL)	Filling technique	Filtration and preservation	Holding time	Notes
metals						
aluminium barium beryllium cadmium chromium cobalt copper lead manganese molybdenum nickel silver tin vanadium zinc	acid washed, plastic or glass	100		acidify with nitric acid to pH 1 to 2	1 month	
antimony	acid washed, plastic or glass	100		acidify with nitric acid or hydrochloric acid to pH 1 to 2	1 month	hydrochloric acid should be used if hydride technique is used for analysis—consult laboratory
arsenic	acid washed, plastic or glass	500	fill container completely to exclude air	acidify with nitric acid or hydrochloric acid to pH 1 to 2	1 month	hydrochloric acid should be used if hydride technique is used for analysis—consult laboratory
boron	plastic	100	fill container completely to exclude air	none required	1 month	
chromium (VI)	acid washed, plastic or glass	100	fill container completely to exclude air	refrigerate	1 day	sample container should be rinsed thoroughly

Analyte	Container type	Typical volume (mL)	Filling technique	Filtration and preservation	Holding time	Notes
iron (II)	acid washed, plastic or glass	500	fill container completely to exclude air	acidify with hydrochloric acid to pH 1 to 2	24 hours	
iron, total	acid washed, plastic or glass	500		acidify with nitric acid to pH 1 to 2	1 month	
lithium	plastic	100		none required, but may acidify with nitric acid to pH 1 to 2 and refrigerate	1 month	acidification allows the sample to be analysed for lithium as well as other metals
magnesium	acid washed, plastic or glass	100	fill container completely to exclude air	none required	1 week	samples with pH > 8 or high carbonate content to be analysed solely for calcium, magnesium or hardness should be acidified
				acidify with nitric acid to pH 1 to 2	1 month	acidification permits determination of other metals from same sample
mercury	acid washed, glass	500		acidify with nitric acid to pH 1 to 2 and add potassium dichromate to give a 0.05% (m/v) final concentration	1 month	particular care is needed to ensure that the sample containers are free from contamination
potassium	acid washed, plastic or glass	100		none required/acidify with nitric acid to pH 1 to 2	1 month	acidification allows the sample to be analysed for potassium as well as other metals
selenium	acid washed, plastic or glass	500		acidify with nitric or hydrochloric acid to pH 1 to 2	1 month	
uranium	acid washed, plastic or glass	200		acidify with nitric acid to pH 1 to 2	1 month	
inorganic (non-metallic)						

Appendix F: PALABOS code for Cascade Aerator

damBreak3d_1

```
/* This file is part of the Palabos library.
 *
 * Copyright (C) 2011-2015 FlowKit Sarl
 * Route d'Oron 2
 * 1010 Lausanne, Switzerland
 * E-mail contact: contact@flowkit.com
 *
 * The most recent release of Palabos can be downloaded at
 * <http://www.palabos.org/>
 *
 * The library Palabos is free software: you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * The library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The breaking dam free surface problem. This code demonstrates the basic usage of the
 * free surface module in Palabos. Surface tension and contact angles are optional.
 */

#include "palabos3D.h"
#include "palabos3D.hh"

using namespace plb;
using namespace std;

#define DESCRIPTOR descriptors::ForcedD3Q19Descriptor
typedef double T;


// Smagorinsky constant for LES model.
const T cSmago = 0.14;

// Physical dimensions of the system (in meters).
const T lx = 2.0;
const T ly = 0.3;
const T lz = 1.3;

const T rhoEmpty = T(1);

plint writeImagesIter   = 100;
plint getStatisticsIter = 20;

plint maxIter;
plint N;
plint nx, ny, nz;
T delta_t, delta_x;
Array<T,3> externalForce;
T nuPhys, nuLB, tau, omega, Bo, surfaceTensionLB, contactAngle;

std::string outDir;
plint obstacleCenterXYplane, obstacleLength, obstacleWidth, obstacleHeight, beginWaterReservoir,
waterReservoirHeight;
plint waterLevelOne, waterLevelTwo, waterLevelThree, waterLevelFour;

void setupParameters() {
    delta_x = lz / N;
    nx = util::roundToInt(lx / delta_x);
```

```
    ny = util::roundToInt(ly / delta_x);
    nz = util::roundToInt(lz / delta_x);

    // Gravity in lattice units.
    T gLB = 9.8 * delta_t * delta_t/delta_x;
    externalForce = Array<T,3>(0., 0., -gLB);
    tau            = (nuPhys*DESCRIPTOR<T>::invCs2*delta_t)/(delta_x*delta_x) + 0.5;
    omega          = 1./tau;
    nuLB           = (tau-0.5)*DESCRIPTOR<T>::cs2; // Viscosity in lattice units.

    surfaceTensionLB = rhoEmpty * gLB * N * N / Bo;

    obstacleCenterXYplane = util::roundToInt(0.744*N);
    obstacleLength        = util::roundToInt(0.403*N);
    obstacleWidth         = util::roundToInt(0.161*N);
    obstacleHeight        = util::roundToInt(0.161*N);
    beginWaterReservoir   = util::roundToInt((1.43/2.0)*nx);
    waterReservoirHeight  = util::roundToInt(nz);

    waterLevelOne   = util::roundToInt(0.496*N);
    waterLevelTwo   = util::roundToInt(2.*0.496*N);
    waterLevelThree = util::roundToInt(3.*0.496*N);
    waterLevelFour  = util::roundToInt((3.*0.496 + 1.150)*N);
}

// Specifies the initial condition for the fluid (each cell is assigned the
// flag "fluid", "empty", or "wall").
int initialFluidFlags(plint iX, plint iY, plint iZ) {
    // Place an obstacle on the left end, which is hit by the fluid.
        bool insideObstacle =
        iX >= (1.4/2.0)*nx &&
        iX <= (1.43/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.74/1.3)*nz;

bool insideObstacle1 =
        iX >= (1.1/2.0)*nx &&
        iX <= (1.40/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.64/1.3)*nz;

bool insideObstacle2 =
        iX >= (0.8/2.0)*nx &&
        iX <= (1.1/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.48/1.3)*nz;

bool insideObstacle3 =
        iX >= (0.5/2.0)*nx &&
        iX <= (0.8/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.32/1.3)*nz;

bool insideObstacle4 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.5/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
```

```
        iZ >=  0 &&
        iZ <= (0.16/1.3)*nz;

bool insideObstacle5 =
        iX >=  0 &&
        iX <= (0.2/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.04/1.3)*nz;

bool insideObstacle6 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.21/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >= (0.16/1.3)*nz &&
        iZ <= (0.182/1.3)*nz;

bool insideObstacle7 =
        iX >=  (0.5/2.0)*nx &&
        iX <=  (0.51/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.32/1.3)*nz &&
        iZ <=  (0.342/1.3)*nz;

bool insideObstacle8 =
        iX >=  (0.8/2.0)*nx &&
        iX <=  (0.81/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.48/1.3)*nz &&
        iZ <=  (0.502/1.3)*nz;

bool insideObstacle9 =
        iX >=  (1.1/2.0)*nx &&
        iX <=  (1.11/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.64/1.3)*nz &&
        iZ <=  (0.662/1.3)*nz;

bool insideObstacle10 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  0 &&
        iY <=  (0.1/0.3)*ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle11 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  (0.2/0.3)*ny &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle12 =
        iX >=  (1.8/2.0)*nx &&
        iX <=  (1.85/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.2/1.3)*nz &&
        iZ <=  nz;
```

```
198
199     if (insideObstacle) {
200         return twoPhaseFlag::wall;
201     }
202     else if (insideObstacle1) {
203         return twoPhaseFlag::wall;
204     }
205     else if (insideObstacle2) {
206         return twoPhaseFlag::wall;
207     }
208     else if (insideObstacle3) {
209         return twoPhaseFlag::wall;
210     }
211     else if (insideObstacle4) {
212         return twoPhaseFlag::wall;
213     }
214     else if (insideObstacle5) {
215         return twoPhaseFlag::wall;
216     }
217     else if (insideObstacle6) {
218         return twoPhaseFlag::wall;
219     }
220     else if (insideObstacle7) {
221         return twoPhaseFlag::wall;
222     }
223     else if (insideObstacle8) {
224         return twoPhaseFlag::wall;
225     }
226     else if (insideObstacle9) {
227         return twoPhaseFlag::wall;
228     }
229     else if (insideObstacle10) {
230         return twoPhaseFlag::wall;
231     }
232     else if (insideObstacle11) {
233         return twoPhaseFlag::wall;
234     }
235     else if (insideObstacle12) {
236         return twoPhaseFlag::wall;
237     }
238
239     else if (iX >= beginWaterReservoir && iZ <= waterReservoirHeight) {
240         return twoPhaseFlag::fluid;
241     }
242     else {
243         return twoPhaseFlag::empty;
244     }
245 }
246
247 void writeResults(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, MultiScalarField3D<T>& volumeFraction, plint
iT)
248 {
249     static const plint nx = lattice.getNx();
250     static const plint ny = lattice.getNy();
251     static const plint nz = lattice.getNz();
252     Box3D slice(0, nx-1, ny/2, ny/2, 0, nz-1);
253     ImageWriter<T> imageWriter("leeloo");
254     imageWriter.writeScaledPpm(createFileName("u", iT, 6),
255                                *computeVelocityNorm(lattice, slice));
256
257     imageWriter.writeScaledPpm(createFileName("rho", iT, 6),
258                                *computeDensity(lattice, slice));
259
260 //     imageWriter.writeScaledPpm(createFileName("p", iT, 6),
261 //                                computePressure(lattice, slice));
262
```

```
263       imageWriter.writeScaledPpm(createFileName("volumeFraction", iT, 6), *extractSubDomain(volumeFraction,
slice));
264
265       // Use a marching-cube algorithm to reconstruct the free surface and write an STL file.
266       std::vector<T> isoLevels;
267       isoLevels.push_back((T) 0.5);
268       typedef TriangleSet<T>::Triangle Triangle;
269       std::vector<Triangle> triangles;
270       isoSurfaceMarchingCube(triangles, volumeFraction, isoLevels, volumeFraction.getBoundingBox());
271       TriangleSet<T>(triangles).writeBinarySTL(createFileName(outDir+"/interface", iT, 6)+".stl");
272
273       VtkImageOutput3D<T> vtkOut(createFileName("volumeFraction", iT, 6), 1.);
274       vtkOut.writeData<float>(volumeFraction, "vf", 1.);
275   }
276
277   //void writeStatistics(FreeSurfaceFields3D<T,DESCRIPTOR>& fields) {
278   void writeStatistics(TwoPhaseFields3D<T,DESCRIPTOR>& fields) {
279       pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
280       T averageMass = freeSurfaceAverageMass<T,DESCRIPTOR>(fields.twoPhaseArgs, fields.lattice.getBoundingBox
());
281       pcout << "Average Mass: " << averageMass  << endl;
282       T averageDensity = freeSurfaceAverageDensity<T,DESCRIPTOR>(fields.twoPhaseArgs, fields.lattice.
getBoundingBox());
283       pcout << "Average Density: " << setprecision(12) << averageDensity  << endl;
284
285       T averageVolumeFraction = freeSurfaceAverageVolumeFraction<T,DESCRIPTOR>(fields.twoPhaseArgs, fields.
lattice.getBoundingBox());
286       pcout << "Average Volume-Fraction: " << setprecision(12) << averageVolumeFraction  << endl;
287
288       pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
289   }
290
291   void writeResultsP(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
292   {
293       static const plint nx = lattice.getNx();
294       static const plint ny = lattice.getNy();
295       static const plint nz = lattice.getNz();
296       Box3D boundingBoxP(0, nx-1, 0, ny-1, 0, nz-1);
297
298       VtkImageOutput3D<T>vtkOutP(createFileName("Pressure", iT, 8), 1.);
299       std::auto_ptr<MultiScalarField3D<T> > px = fields->computePressure(boundingBoxP);
300       vtkOutP.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
301
302   //      std::auto_ptr<MultiTensorField3D<T> > vx = fields->computeVelocity(boundingBoxP);
303   //      vtkOutPV.writeData<3,float>(*vx, "v", delta_x / delta_t);
304
305   }
306
307
308   void writeResultsV(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
309   {
310       static const plint nx = lattice.getNx();
311       static const plint ny = lattice.getNy();
312       static const plint nz = lattice.getNz();
313       Box3D boundingBoxV(0, nx-1, 0, ny-1, 0, nz-1);
314
315       VtkImageOutput3D<T>vtkOutV(createFileName("Velocity", iT, 8), 1.);
316   //      std::auto_ptr<MultiScalarField3D<T>> px = fields->computePressure(boundingBoxP);
317   //      vtkOutPV.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
318
319       std::auto_ptr<MultiTensorField3D<T,3> > vx = fields->computeVelocity(boundingBoxV);
320       vtkOutV.writeData<3,float>(*vx, "v", delta_x / delta_t);
321
322   }
```

```
323
324 int main(int argc, char **argv)
325 {
326     plbInit(&argc, &argv);
327     global::directories().setInputDir("./");
328
329     if (global::argc() != 8) {
330         pcout << "Error missing some input parameter\n";
331     }
332
333     try {
334         global::argv(1).read(outDir);
335         global::directories().setOutputDir(outDir+"/");
336
337         global::argv(2).read(nuPhys);
338         global::argv(3).read(Bo);
339         global::argv(4).read(contactAngle);
340         global::argv(5).read(N);
341         global::argv(6).read(delta_t);
342         global::argv(7).read(maxIter);
343     }
344     catch(PlbIOException& except) {
345         pcout << except.what() << std::endl;
346         pcout << "The parameters for this program are :\n";
347         pcout << "1. Output directory name.\n";
348         pcout << "2. kinematic viscosity in physical Units (m^2/s) .\n";
349         pcout << "3. Bond number (Bo = rho * g * L^2 / gamma).\n";
350         pcout << "4. Contact angle (in degrees).\n";
351         pcout << "5. number of lattice nodes for lz .\n";
352         pcout << "6. delta_t .\n";
353         pcout << "7. maxIter .\n";
354         pcout << "Reasonable parameters on a desktop computer are: " << (std::string)global::argv(0) << "
tmp 1.e-5 100 80.0 40 1.e-3 80000\n";
355         pcout << "Reasonable parameters on a parallel machine are: " << (std::string)global::argv(0) << "
tmp 1.e-6 100 80.0 100 1.e-4 80000\n";
356         exit (EXIT_FAILURE);
357     }
358
359     setupParameters();
360
361     pcout << "delta_t= " << delta_t << endl;
362     pcout << "delta_x= " << delta_x << endl;
363     pcout << "delta_t*delta_t/delta_x= " << delta_t*delta_t/delta_x << endl;
364     pcout << "externalForce= " << externalForce[2] << endl;
365     pcout << "relaxation time= " << tau << endl;
366     pcout << "omega= " << omega << endl;
367     pcout << "kinematic viscosity physical units = " << nuPhys << endl;
368     pcout << "kinematic viscosity lattice units= " << nuLB << endl;
369
370     global::timer("initialization").start();
371
372
373     SparseBlockStructure3D blockStructure(createRegularDistribution3D(nx, ny, nz));
374
375     Dynamics<T,DESCRIPTOR>* dynamics
376         = new SmagorinskyBGKdynamics<T,DESCRIPTOR>(omega, cSmago);
377
378     // If surfaceTensionLB is 0, then the surface tension algorithm is deactivated.
379     // If contactAngle is less than 0, then the contact angle algorithm is deactivated.
380     TwoPhaseFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
381                                            surfaceTensionLB, contactAngle, externalForce );
382     //FreeSurfaceFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
383     //                                          surfaceTensionLB, contactAngle, externalForce, false );
384     //integrateProcessingFunctional(new ShortenBounceBack3D<T,DESCRIPTOR>, fields.lattice.getBoundingBox(),
fields.twoPhaseArgs, 0);
385
```

```
386     // Set all outer-wall cells to "wall" (here, bulk-cells are also set to "wall", but it
387     // doesn't matter, because they are overwritten on the next line).
388     setToConstant(fields.flag, fields.flag.getBoundingBox(), (int)twoPhaseFlag::wall);
389     // In the bulk (all except outer wall layer), initialize the flags as specified by
390     // the function "initialFluidFlags".
391     setToFunction(fields.flag, fields.flag.getBoundingBox().enlarge(-1), initialFluidFlags);
392
393     fields.defaultInitialize();
394
395     pcout << "Time spent for setting up lattices: "
396           << global::timer("initialization").stop() << endl;
397     T lastIterationTime = T();
398
399     for (plint iT = 0; iT <= maxIter; ++iT) {
400         global::timer("iteration").restart();
401
402         T sum_of_mass_matrix = T();
403         T lost_mass = T();
404         if (iT % getStatisticsIter==0) {
405             pcout << endl;
406             pcout << "ITERATION = " << iT << endl;
407             pcout << "Time of last iteration is " << lastIterationTime << " seconds" << endl;
408             writeStatistics(fields);
409             sum_of_mass_matrix = fields.lattice.getInternalStatistics().getSum(0);
410             pcout << "Sum of mass matrix: " << sum_of_mass_matrix << std::endl;
411             lost_mass = fields.lattice.getInternalStatistics().getSum(1);
412             pcout << "Lost mass: " << lost_mass << std::endl;
413             pcout << "Total mass: " << sum_of_mass_matrix + lost_mass << std::endl;
414             pcout << "Interface cells: " << fields.lattice.getInternalStatistics().getIntSum(0) << std::
endl;
415         }
416
417         if (iT % writeImagesIter == 0) {
418             global::timer("images").start();
419             writeResultsP(fields.lattice,&fields, iT);   //added for pressure results output
420             writeResultsV(fields.lattice,&fields, iT);   //added for velocity results output
421             writeResults(fields.lattice, fields.volumeFraction, iT);
422             pcout << "Total time spent for writing images: "
423                 << global::timer("images").stop() << endl;
424         }
425
426         // This includes the collision-streaming cycle, plus all free-surface operations.
427         fields.lattice.executeInternalProcessors();
428         fields.lattice.evaluateStatistics();
429         fields.lattice.incrementTime();
430
431         lastIterationTime = global::timer("iteration").stop();
432     }
433 }
434
```

damBreak3d_2

```
/* This file is part of the Palabos library.
 *
 * Copyright (C) 2011-2015 FlowKit Sarl
 * Route d'Oron 2
 * 1010 Lausanne, Switzerland
 * E-mail contact: contact@flowkit.com
 *
 * The most recent release of Palabos can be downloaded at
 * <http://www.palabos.org/>
 *
 * The library Palabos is free software: you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * The library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The breaking dam free surface problem. This code demonstrates the basic usage of the
 * free surface module in Palabos. Surface tension and contact angles are optional.
 */

#include "palabos3D.h"
#include "palabos3D.hh"

using namespace plb;
using namespace std;

#define DESCRIPTOR descriptors::ForcedD3Q19Descriptor
typedef double T;


// Smagorinsky constant for LES model.
const T cSmago = 0.14;

// Physical dimensions of the system (in meters).
const T lx = 2.0;
const T ly = 0.3;
const T lz = 1.3;

const T rhoEmpty = T(1);

plint writeImagesIter   = 100;
plint getStatisticsIter = 20;

plint maxIter;
plint N;
plint nx, ny, nz;
T delta_t, delta_x;
Array<T,3> externalForce;
T nuPhys, nuLB, tau, omega, Bo, surfaceTensionLB, contactAngle;

std::string outDir;
plint obstacleCenterXYplane, obstacleLength, obstacleWidth, obstacleHeight, beginWaterReservoir,
waterReservoirHeight;
plint waterLevelOne, waterLevelTwo, waterLevelThree, waterLevelFour;

void setupParameters() {
    delta_x = lz / N;
    nx = util::roundToInt(lx / delta_x);
```

```
    ny = util::roundToInt(ly / delta_x);
    nz = util::roundToInt(lz / delta_x);

    // Gravity in lattice units.
    T gLB = 9.8 * delta_t * delta_t/delta_x;
    externalForce = Array<T,3>(0., 0., -gLB);
    tau           = (nuPhys*DESCRIPTOR<T>::invCs2*delta_t)/(delta_x*delta_x) + 0.5;
    omega         = 1./tau;
    nuLB          = (tau-0.5)*DESCRIPTOR<T>::cs2; // Viscosity in lattice units.

    surfaceTensionLB = rhoEmpty * gLB * N * N / Bo;

    obstacleCenterXYplane = util::roundToInt(0.744*N);
    obstacleLength        = util::roundToInt(0.403*N);
    obstacleWidth         = util::roundToInt(0.161*N);
    obstacleHeight        = util::roundToInt(0.161*N);
    beginWaterReservoir   = util::roundToInt((1.43/2.0)*nx);
    waterReservoirHeight  = util::roundToInt(nz);

    waterLevelOne   = util::roundToInt(0.496*N);
    waterLevelTwo   = util::roundToInt(2.*0.496*N);
    waterLevelThree = util::roundToInt(3.*0.496*N);
    waterLevelFour  = util::roundToInt((3.*0.496 + 1.150)*N);
}

// Specifies the initial condition for the fluid (each cell is assigned the
// flag "fluid", "empty", or "wall").
int initialFluidFlags(plint iX, plint iY, plint iZ) {
    // Place an obstacle on the left end, which is hit by the fluid.
        bool insideObstacle =
        iX >= (1.4/2.0)*nx &&
        iX <= (1.43/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.556/1.3)*nz;

bool insideObstacle1 =
        iX >= (1.1/2.0)*nx &&
        iX <= (1.40/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.456/1.3)*nz;

bool insideObstacle2 =
        iX >= (0.8/2.0)*nx &&
        iX <= (1.1/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.342/1.3)*nz;

bool insideObstacle3 =
        iX >= (0.5/2.0)*nx &&
        iX <= (0.8/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.228/1.3)*nz;

bool insideObstacle4 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.5/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
```

```
        iZ >=  0 &&
        iZ <= (0.114/1.3)*nz;

bool insideObstacle5 =
        iX >=  0 &&
        iX <= (0.2/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.029/1.3)*nz;

bool insideObstacle6 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.21/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >= (0.114/1.3)*nz &&
        iZ <= (0.136/1.3)*nz;

bool insideObstacle7 =
        iX >=  (0.5/2.0)*nx &&
        iX <=  (0.51/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.228/1.3)*nz &&
        iZ <=  (0.25/1.3)*nz;

bool insideObstacle8 =
        iX >=  (0.8/2.0)*nx &&
        iX <=  (0.81/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.342/1.3)*nz &&
        iZ <=  (0.364/1.3)*nz;

bool insideObstacle9 =
        iX >=  (1.1/2.0)*nx &&
        iX <=  (1.11/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.456/1.3)*nz &&
        iZ <=  (0.478/1.3)*nz;

bool insideObstacle10 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  0 &&
        iY <=  (0.1/0.3)*ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle11 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  (0.2/0.3)*ny &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle12 =
        iX >=  (1.8/2.0)*nx &&
        iX <=  (1.85/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.2/1.3)*nz &&
        iZ <=  nz;
```

```
198
199     if (insideObstacle) {
200         return twoPhaseFlag::wall;
201     }
202     else if (insideObstacle1) {
203         return twoPhaseFlag::wall;
204     }
205     else if (insideObstacle2) {
206         return twoPhaseFlag::wall;
207     }
208     else if (insideObstacle3) {
209         return twoPhaseFlag::wall;
210     }
211     else if (insideObstacle4) {
212         return twoPhaseFlag::wall;
213     }
214     else if (insideObstacle5) {
215         return twoPhaseFlag::wall;
216     }
217     else if (insideObstacle6) {
218         return twoPhaseFlag::wall;
219     }
220     else if (insideObstacle7) {
221         return twoPhaseFlag::wall;
222     }
223     else if (insideObstacle8) {
224         return twoPhaseFlag::wall;
225     }
226     else if (insideObstacle9) {
227         return twoPhaseFlag::wall;
228     }
229     else if (insideObstacle10) {
230         return twoPhaseFlag::wall;
231     }
232     else if (insideObstacle11) {
233         return twoPhaseFlag::wall;
234     }
235     else if (insideObstacle12) {
236         return twoPhaseFlag::wall;
237     }
238
239     else if (iX >= beginWaterReservoir && iZ <= waterReservoirHeight) {
240         return twoPhaseFlag::fluid;
241     }
242     else {
243         return twoPhaseFlag::empty;
244     }
245 }
246
247 void writeResults(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, MultiScalarField3D<T>& volumeFraction, plint
iT)
248 {
249     static const plint nx = lattice.getNx();
250     static const plint ny = lattice.getNy();
251     static const plint nz = lattice.getNz();
252     Box3D slice(0, nx-1, ny/2, ny/2, 0, nz-1);
253     ImageWriter<T> imageWriter("leeloo");
254     imageWriter.writeScaledPpm(createFileName("u", iT, 6),
255                                *computeVelocityNorm(lattice, slice));
256
257     imageWriter.writeScaledPpm(createFileName("rho", iT, 6),
258                                *computeDensity(lattice, slice));
259
260     imageWriter.writeScaledPpm(createFileName("volumeFraction", iT, 6), *extractSubDomain(volumeFraction,
slice));
261
```

```
262       // Use a marching-cube algorithm to reconstruct the free surface and write an STL file.
263       std::vector<T> isoLevels;
264       isoLevels.push_back((T) 0.5);
265       typedef TriangleSet<T>::Triangle Triangle;
266       std::vector<Triangle> triangles;
267       isoSurfaceMarchingCube(triangles, volumeFraction, isoLevels, volumeFraction.getBoundingBox());
268       TriangleSet<T>(triangles).writeBinarySTL(createFileName(outDir+"/interface", iT, 6)+".stl");
269
270       VtkImageOutput3D<T> vtkOut(createFileName("volumeFraction", iT, 6), 1.);
271       vtkOut.writeData<float>(volumeFraction, "vf", 1.);
272   }
273
274   //void writeStatistics(FreeSurfaceFields3D<T,DESCRIPTOR>& fields) {
275   void writeStatistics(TwoPhaseFields3D<T,DESCRIPTOR>& fields) {
276       pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
277       T averageMass = freeSurfaceAverageMass<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
278       pcout << "Average Mass: " << averageMass  << endl;
279       T averageDensity = freeSurfaceAverageDensity<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
280       pcout << "Average Density: " << setprecision(12) << averageDensity  << endl;
281
282       T averageVolumeFraction = freeSurfaceAverageVolumeFraction<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
283       pcout << "Average Volume-Fraction: " << setprecision(12) << averageVolumeFraction  << endl;
284
285       pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
286   }
287
288   void writeResultsP(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
289   {
290       static const plint nx = lattice.getNx();
291       static const plint ny = lattice.getNy();
292       static const plint nz = lattice.getNz();
293       Box3D boundingBoxP(0, nx-1, 0, ny-1, 0, nz-1);
294
295       VtkImageOutput3D<T>vtkOutP(createFileName("Pressure", iT, 8), 1.);
296       std::auto_ptr<MultiScalarField3D<T> > px = fields->computePressure(boundingBoxP);
297       vtkOutP.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
298
299   //     std::auto_ptr<MultiTensorField3D<T> > vx = fields->computeVelocity(boundingBoxP);
300   //     vtkOutPV.writeData<3,float>(*vx, "v", delta_x / delta_t);
301
302   }
303
304
305   void writeResultsV(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
306   {
307       static const plint nx = lattice.getNx();
308       static const plint ny = lattice.getNy();
309       static const plint nz = lattice.getNz();
310       Box3D boundingBoxV(0, nx-1, 0, ny-1, 0, nz-1);
311
312       VtkImageOutput3D<T>vtkOutV(createFileName("Velocity", iT, 8), 1.);
313   //     std::auto_ptr<MultiScalarField3D<T>> px = fields->computePressure(boundingBoxP);
314   //     vtkOutPV.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
315
316       std::auto_ptr<MultiTensorField3D<T,3> > vx = fields->computeVelocity(boundingBoxV);
317       vtkOutV.writeData<3,float>(*vx, "v", delta_x / delta_t);
318
319   }
320
321   int main(int argc, char **argv)
322   {
```

```
323     plbInit(&argc, &argv);
324     global::directories().setInputDir("./");
325
326     if (global::argc() != 8) {
327         pcout << "Error missing some input parameter\n";
328     }
329
330     try {
331         global::argv(1).read(outDir);
332         global::directories().setOutputDir(outDir+"/");
333
334         global::argv(2).read(nuPhys);
335         global::argv(3).read(Bo);
336         global::argv(4).read(contactAngle);
337         global::argv(5).read(N);
338         global::argv(6).read(delta_t);
339         global::argv(7).read(maxIter);
340     }
341     catch(PlbIOException& except) {
342         pcout << except.what() << std::endl;
343         pcout << "The parameters for this program are :\n";
344         pcout << "1. Output directory name.\n";
345         pcout << "2. kinematic viscosity in physical Units (m^2/s) .\n";
346         pcout << "3. Bond number (Bo = rho * g * L^2 / gamma).\n";
347         pcout << "4. Contact angle (in degrees).\n";
348         pcout << "5. number of lattice nodes for lz .\n";
349         pcout << "6. delta_t .\n";
350         pcout << "7. maxIter .\n";
351         pcout << "Reasonable parameters on a desktop computer are: " << (std::string)global::argv(0) << "
tmp 1.e-5 100 80.0 40 1.e-3 80000\n";
352         pcout << "Reasonable parameters on a parallel machine are: " << (std::string)global::argv(0) << "
tmp 1.e-6 100 80.0 100 1.e-4 80000\n";
353         exit (EXIT_FAILURE);
354     }
355
356     setupParameters();
357
358     pcout << "delta_t= " << delta_t << endl;
359     pcout << "delta_x= " << delta_x << endl;
360     pcout << "delta_t*delta_t/delta_x= " << delta_t*delta_t/delta_x << endl;
361     pcout << "externalForce= " << externalForce[2] << endl;
362     pcout << "relaxation time= " << tau << endl;
363     pcout << "omega= " << omega << endl;
364     pcout << "kinematic viscosity physical units = " << nuPhys << endl;
365     pcout << "kinematic viscosity lattice units= " << nuLB << endl;
366
367     global::timer("initialization").start();
368
369
370     SparseBlockStructure3D blockStructure(createRegularDistribution3D(nx, ny, nz));
371
372     Dynamics<T,DESCRIPTOR>* dynamics
373         = new SmagorinskyBGKdynamics<T,DESCRIPTOR>(omega, cSmago);
374
375     // If surfaceTensionLB is 0, then the surface tension algorithm is deactivated.
376     // If contactAngle is less than 0, then the contact angle algorithm is deactivated.
377     TwoPhaseFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
378                                            surfaceTensionLB, contactAngle, externalForce );
379     //FreeSurfaceFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
380     //                                          surfaceTensionLB, contactAngle, externalForce, false );
381     //integrateProcessingFunctional(new ShortenBounceBack3D<T,DESCRIPTOR>, fields.lattice.getBoundingBox(),
fields.twoPhaseArgs, 0);
382
383     // Set all outer-wall cells to "wall" (here, bulk-cells are also set to "wall", but it
384     // doesn't matter, because they are overwritten on the next line).
385     setToConstant(fields.flag, fields.flag.getBoundingBox(), (int)twoPhaseFlag::wall);
```

```
386     // In the bulk (all except outer wall layer), initialize the flags as specified by
387     // the function "initialFluidFlags".
388     setToFunction(fields.flag, fields.flag.getBoundingBox().enlarge(-1), initialFluidFlags);
389
390     fields.defaultInitialize();
391
392     pcout << "Time spent for setting up lattices: "
393           << global::timer("initialization").stop() << endl;
394     T lastIterationTime = T();
395
396     for (plint iT = 0; iT <= maxIter; ++iT) {
397         global::timer("iteration").restart();
398
399         T sum_of_mass_matrix = T();
400         T lost_mass = T();
401         if (iT % getStatisticsIter==0) {
402             pcout << endl;
403             pcout << "ITERATION = " << iT << endl;
404             pcout << "Time of last iteration is " << lastIterationTime << " seconds" << endl;
405             writeStatistics(fields);
406             sum_of_mass_matrix = fields.lattice.getInternalStatistics().getSum(0);
407             pcout << "Sum of mass matrix: " << sum_of_mass_matrix << std::endl;
408             lost_mass = fields.lattice.getInternalStatistics().getSum(1);
409             pcout << "Lost mass: " << lost_mass << std::endl;
410             pcout << "Total mass: " << sum_of_mass_matrix + lost_mass << std::endl;
411             pcout << "Interface cells: " << fields.lattice.getInternalStatistics().getIntSum(0) <<
std::endl;
412         }
413
414         if (iT % writeImagesIter == 0) {
415             global::timer("images").start();
416             writeResultsP(fields.lattice,&fields, iT);   //added for pressure results output
417             writeResultsV(fields.lattice,&fields, iT);   //added for velocity results output
418             writeResults(fields.lattice, fields.volumeFraction, iT);
419             pcout << "Total time spent for writing images: "
420                 << global::timer("images").stop() << endl;
421         }
422
423         // This includes the collision-streaming cycle, plus all free-surface operations.
424         fields.lattice.executeInternalProcessors();
425         fields.lattice.evaluateStatistics();
426         fields.lattice.incrementTime();
427
428         lastIterationTime = global::timer("iteration").stop();
429     }
430 }
431
```

damBreak3d_3

```
/* This file is part of the Palabos library.
 *
 * Copyright (C) 2011-2015 FlowKit Sarl
 * Route d'Oron 2
 * 1010 Lausanne, Switzerland
 * E-mail contact: contact@flowkit.com
 *
 * The most recent release of Palabos can be downloaded at
 * <http://www.palabos.org/>
 *
 * The library Palabos is free software: you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * The library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The breaking dam free surface problem. This code demonstrates the basic usage of the
 * free surface module in Palabos. Surface tension and contact angles are optional.
 */

#include "palabos3D.h"
#include "palabos3D.hh"

using namespace plb;
using namespace std;

#define DESCRIPTOR descriptors::ForcedD3Q19Descriptor
typedef double T;


// Smagorinsky constant for LES model.
const T cSmago = 0.14;

// Physical dimensions of the system (in meters).
const T lx = 2.0;
const T ly = 0.3;
const T lz = 1.3;

const T rhoEmpty = T(1);

plint writeImagesIter   = 100;
plint getStatisticsIter = 20;

plint maxIter;
plint N;
plint nx, ny, nz;
T delta_t, delta_x;
Array<T,3> externalForce;
T nuPhys, nuLB, tau, omega, Bo, surfaceTensionLB, contactAngle;

std::string outDir;
plint obstacleCenterXYplane, obstacleLength, obstacleWidth, obstacleHeight, beginWaterReservoir,
waterReservoirHeight;
plint waterLevelOne, waterLevelTwo, waterLevelThree, waterLevelFour;

void setupParameters() {
    delta_x = lz / N;
    nx = util::roundToInt(lx / delta_x);
```

```
    ny = util::roundToInt(ly / delta_x);
    nz = util::roundToInt(lz / delta_x);

    // Gravity in lattice units.
    T gLB = 9.8 * delta_t * delta_t/delta_x;
    externalForce = Array<T,3>(0., 0., -gLB);
    tau            = (nuPhys*DESCRIPTOR<T>::invCs2*delta_t)/(delta_x*delta_x) + 0.5;
    omega          = 1./tau;
    nuLB           = (tau-0.5)*DESCRIPTOR<T>::cs2; // Viscosity in lattice units.

    surfaceTensionLB = rhoEmpty * gLB * N * N / Bo;

    obstacleCenterXYplane = util::roundToInt(0.744*N);
    obstacleLength        = util::roundToInt(0.403*N);
    obstacleWidth         = util::roundToInt(0.161*N);
    obstacleHeight        = util::roundToInt(0.161*N);
    beginWaterReservoir   = util::roundToInt((1.43/2.0)*nx);
    waterReservoirHeight  = util::roundToInt(nz);

    waterLevelOne   = util::roundToInt(0.496*N);
    waterLevelTwo   = util::roundToInt(2.*0.496*N);
    waterLevelThree = util::roundToInt(3.*0.496*N);
    waterLevelFour  = util::roundToInt((3.*0.496 + 1.150)*N);
}

// Specifies the initial condition for the fluid (each cell is assigned the
// flag "fluid", "empty", or "wall").
int initialFluidFlags(plint iX, plint iY, plint iZ) {
    // Place an obstacle on the left end, which is hit by the fluid.
        bool insideObstacle =
        iX >= (1.4/2.0)*nx &&
        iX <= (1.43/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.5/1.3)*nz;

bool insideObstacle1 =
        iX >= (1.1/2.0)*nx &&
        iX <= (1.40/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.4/1.3)*nz;

bool insideObstacle2 =
        iX >= (0.8/2.0)*nx &&
        iX <= (1.1/2.0)*nx &&
        iY >=  0 &&
        iY <= ny &&
        iZ >= 0 &&
        iZ <= (0.3/1.3)*nz;

bool insideObstacle3 =
        iX >= (0.5/2.0)*nx &&
        iX <= (0.8/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.2/1.3)*nz;

bool insideObstacle4 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.5/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
```

```
132         iZ >=  0 &&
133         iZ <= (0.1/1.3)*nz;
134
135 bool insideObstacle5 =
136         iX >=  0 &&
137         iX <= (0.2/2.0)*nx &&
138         iY >=  0 &&
139         iY <=  ny &&
140         iZ >=  0 &&
141         iZ <= (0.025/1.3)*nz;
142
143 bool insideObstacle6 =
144         iX >= (0.2/2.0)*nx &&
145         iX <= (0.21/2.0)*nx &&
146         iY >=  0 &&
147         iY <=  ny &&
148         iZ >= (0.1/1.3)*nz &&
149         iZ <= (0.122/1.3)*nz;
150
151 bool insideObstacle7 =
152         iX >=  (0.5/2.0)*nx &&
153         iX <=  (0.51/2.0)*nx &&
154         iY >=  0 &&
155         iY <=  ny &&
156         iZ >=  (0.2/1.3)*nz &&
157         iZ <=  (0.222/1.3)*nz;
158
159 bool insideObstacle8 =
160         iX >=  (0.8/2.0)*nx &&
161         iX <=  (0.81/2.0)*nx &&
162         iY >=  0 &&
163         iY <=  ny &&
164         iZ >=  (0.3/1.3)*nz &&
165         iZ <=  (0.322/1.3)*nz;
166
167 bool insideObstacle9 =
168         iX >=  (1.1/2.0)*nx &&
169         iX <=  (1.11/2.0)*nx &&
170         iY >=  0 &&
171         iY <=  ny &&
172         iZ >=  (0.4/1.3)*nz &&
173         iZ <=  (0.422/1.3)*nz;
174
175 bool insideObstacle10 =
176         iX >=  0 &&
177         iX <=  (1.4/2.0)*nx &&
178         iY >=  0 &&
179         iY <=  (0.1/0.3)*ny &&
180         iZ >=  0 &&
181         iZ <=  nz;
182
183 bool insideObstacle11 =
184         iX >=  0 &&
185         iX <=  (1.4/2.0)*nx &&
186         iY >=  (0.2/0.3)*ny &&
187         iY <=  ny &&
188         iZ >=  0 &&
189         iZ <=  nz;
190
191 bool insideObstacle12 =
192         iX >=  (1.8/2.0)*nx &&
193         iX <=  (1.85/2.0)*nx &&
194         iY >=  0 &&
195         iY <=  ny &&
196         iZ >=  (0.2/1.3)*nz &&
197         iZ <=  nz;
```

```
198
199     if (insideObstacle) {
200         return twoPhaseFlag::wall;
201     }
202     else if (insideObstacle1) {
203         return twoPhaseFlag::wall;
204     }
205     else if (insideObstacle2) {
206         return twoPhaseFlag::wall;
207     }
208     else if (insideObstacle3) {
209         return twoPhaseFlag::wall;
210     }
211     else if (insideObstacle4) {
212         return twoPhaseFlag::wall;
213     }
214     else if (insideObstacle5) {
215         return twoPhaseFlag::wall;
216     }
217     else if (insideObstacle6) {
218         return twoPhaseFlag::wall;
219     }
220     else if (insideObstacle7) {
221         return twoPhaseFlag::wall;
222     }
223     else if (insideObstacle8) {
224         return twoPhaseFlag::wall;
225     }
226     else if (insideObstacle9) {
227         return twoPhaseFlag::wall;
228     }
229     else if (insideObstacle10) {
230         return twoPhaseFlag::wall;
231     }
232     else if (insideObstacle11) {
233         return twoPhaseFlag::wall;
234     }
235     else if (insideObstacle12) {
236         return twoPhaseFlag::wall;
237     }
238
239     else if (iX >= beginWaterReservoir && iZ <= waterReservoirHeight) {
240         return twoPhaseFlag::fluid;
241     }
242     else {
243         return twoPhaseFlag::empty;
244     }
245 }
246
247 void writeResults(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, MultiScalarField3D<T>& volumeFraction, plint
iT)
248 {
249     static const plint nx = lattice.getNx();
250     static const plint ny = lattice.getNy();
251     static const plint nz = lattice.getNz();
252     Box3D slice(0, nx-1, ny/2, ny/2, 0, nz-1);
253     ImageWriter<T> imageWriter("leeloo");
254     imageWriter.writeScaledPpm(createFileName("u", iT, 6),
255                                *computeVelocityNorm(lattice, slice));
256
257     imageWriter.writeScaledPpm(createFileName("rho", iT, 6),
258                                *computeDensity(lattice, slice));
259
260     imageWriter.writeScaledPpm(createFileName("volumeFraction", iT, 6), *extractSubDomain(volumeFraction,
slice));
261
```

```
262      // Use a marching-cube algorithm to reconstruct the free surface and write an STL file.
263      std::vector<T> isoLevels;
264      isoLevels.push_back((T) 0.5);
265      typedef TriangleSet<T>::Triangle Triangle;
266      std::vector<Triangle> triangles;
267      isoSurfaceMarchingCube(triangles, volumeFraction, isoLevels, volumeFraction.getBoundingBox());
268      TriangleSet<T>(triangles).writeBinarySTL(createFileName(outDir+"/interface", iT, 6)+".stl");
269
270      VtkImageOutput3D<T> vtkOut(createFileName("volumeFraction", iT, 6), 1.);
271      vtkOut.writeData<float>(volumeFraction, "vf", 1.);
272  }
273
274  //void writeStatistics(FreeSurfaceFields3D<T,DESCRIPTOR>& fields) {
275  void writeStatistics(TwoPhaseFields3D<T,DESCRIPTOR>& fields) {
276      pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
277      T averageMass = freeSurfaceAverageMass<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
278      pcout << "Average Mass: " << averageMass  << endl;
279      T averageDensity = freeSurfaceAverageDensity<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
280      pcout << "Average Density: " << setprecision(12) << averageDensity  << endl;
281
282      T averageVolumeFraction = freeSurfaceAverageVolumeFraction<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
283      pcout << "Average Volume-Fraction: " << setprecision(12) << averageVolumeFraction  << endl;
284
285      pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
286  }
287
288  void writeResultsP(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
289  {
290      static const plint nx = lattice.getNx();
291      static const plint ny = lattice.getNy();
292      static const plint nz = lattice.getNz();
293      Box3D boundingBoxP(0, nx-1, 0, ny-1, 0, nz-1);
294
295      VtkImageOutput3D<T>vtkOutP(createFileName("Pressure", iT, 8), 1.);
296      std::auto_ptr<MultiScalarField3D<T> > px = fields->computePressure(boundingBoxP);
297      vtkOutP.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
298
299  //     std::auto_ptr<MultiTensorField3D<T> > vx = fields->computeVelocity(boundingBoxP);
300  //     vtkOutPV.writeData<3,float>(*vx, "v", delta_x / delta_t);
301
302  }
303
304
305  void writeResultsV(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
306  {
307      static const plint nx = lattice.getNx();
308      static const plint ny = lattice.getNy();
309      static const plint nz = lattice.getNz();
310      Box3D boundingBoxV(0, nx-1, 0, ny-1, 0, nz-1);
311
312      VtkImageOutput3D<T>vtkOutV(createFileName("Velocity", iT, 8), 1.);
313  //     std::auto_ptr<MultiScalarField3D<T>> px = fields->computePressure(boundingBoxP);
314  //     vtkOutPV.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
315
316      std::auto_ptr<MultiTensorField3D<T,3> > vx = fields->computeVelocity(boundingBoxV);
317      vtkOutV.writeData<3,float>(*vx, "v", delta_x / delta_t);
318
319  }
320
321  int main(int argc, char **argv)
322  {
```

```
    plbInit(&argc, &argv);
    global::directories().setInputDir("./");

    if (global::argc() != 8) {
        pcout << "Error missing some input parameter\n";
    }

    try {
        global::argv(1).read(outDir);
        global::directories().setOutputDir(outDir+"/");

        global::argv(2).read(nuPhys);
        global::argv(3).read(Bo);
        global::argv(4).read(contactAngle);
        global::argv(5).read(N);
        global::argv(6).read(delta_t);
        global::argv(7).read(maxIter);
    }
    catch(PlbIOException& except) {
        pcout << except.what() << std::endl;
        pcout << "The parameters for this program are :\n";
        pcout << "1. Output directory name.\n";
        pcout << "2. kinematic viscosity in physical Units (m^2/s) .\n";
        pcout << "3. Bond number (Bo = rho * g * L^2 / gamma).\n";
        pcout << "4. Contact angle (in degrees).\n";
        pcout << "5. number of lattice nodes for lz .\n";
        pcout << "6. delta_t .\n";
        pcout << "7. maxIter .\n";
        pcout << "Reasonable parameters on a desktop computer are: " << (std::string)global::argv(0) << "
tmp 1.e-5 100 80.0 40 1.e-3 80000\n";
        pcout << "Reasonable parameters on a parallel machine are: " << (std::string)global::argv(0) << "
tmp 1.e-6 100 80.0 100 1.e-4 80000\n";
        exit (EXIT_FAILURE);
    }

    setupParameters();

    pcout << "delta_t= " << delta_t << endl;
    pcout << "delta_x= " << delta_x << endl;
    pcout << "delta_t*delta_t/delta_x= " << delta_t*delta_t/delta_x << endl;
    pcout << "externalForce= " << externalForce[2] << endl;
    pcout << "relaxation time= " << tau << endl;
    pcout << "omega= " << omega << endl;
    pcout << "kinematic viscosity physical units = " << nuPhys << endl;
    pcout << "kinematic viscosity lattice units= " << nuLB << endl;

    global::timer("initialization").start();


    SparseBlockStructure3D blockStructure(createRegularDistribution3D(nx, ny, nz));

    Dynamics<T,DESCRIPTOR>* dynamics
        = new SmagorinskyBGKdynamics<T,DESCRIPTOR>(omega, cSmago);

    // If surfaceTensionLB is 0, then the surface tension algorithm is deactivated.
    // If contactAngle is less than 0, then the contact angle algorithm is deactivated.
    TwoPhaseFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
                                           surfaceTensionLB, contactAngle, externalForce );
    //FreeSurfaceFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
    //                                          surfaceTensionLB, contactAngle, externalForce, false );
    //integrateProcessingFunctional(new ShortenBounceBack3D<T,DESCRIPTOR>, fields.lattice.getBoundingBox(),
fields.twoPhaseArgs, 0);

    // Set all outer-wall cells to "wall" (here, bulk-cells are also set to "wall", but it
    // doesn't matter, because they are overwritten on the next line).
    setToConstant(fields.flag, fields.flag.getBoundingBox(), (int)twoPhaseFlag::wall);
```

```
386     // In the bulk (all except outer wall layer), initialize the flags as specified by
387     // the function "initialFluidFlags".
388     setToFunction(fields.flag, fields.flag.getBoundingBox().enlarge(-1), initialFluidFlags);
389
390     fields.defaultInitialize();
391
392     pcout << "Time spent for setting up lattices: "
393           << global::timer("initialization").stop() << endl;
394     T lastIterationTime = T();
395
396     for (plint iT = 0; iT <= maxIter; ++iT) {
397         global::timer("iteration").restart();
398
399         T sum_of_mass_matrix = T();
400         T lost_mass = T();
401         if (iT % getStatisticsIter==0) {
402             pcout << endl;
403             pcout << "ITERATION = " << iT << endl;
404             pcout << "Time of last iteration is " << lastIterationTime << " seconds" << endl;
405             writeStatistics(fields);
406             sum_of_mass_matrix = fields.lattice.getInternalStatistics().getSum(0);
407             pcout << "Sum of mass matrix: " << sum_of_mass_matrix << std::endl;
408             lost_mass = fields.lattice.getInternalStatistics().getSum(1);
409             pcout << "Lost mass: " << lost_mass << std::endl;
410             pcout << "Total mass: " << sum_of_mass_matrix + lost_mass << std::endl;
411             pcout << "Interface cells: " << fields.lattice.getInternalStatistics().getIntSum(0) <<
std::endl;
412         }
413
414         if (iT % writeImagesIter == 0) {
415             global::timer("images").start();
416             writeResultsP(fields.lattice,&fields, iT);   //added for pressure results output
417             writeResultsV(fields.lattice,&fields, iT);   //added for velocity results output
418
419             writeResults(fields.lattice, fields.volumeFraction, iT);
420             pcout << "Total time spent for writing images: "
421                 << global::timer("images").stop() << endl;
422         }
423
424         // This includes the collision-streaming cycle, plus all free-surface operations.
425         fields.lattice.executeInternalProcessors();
426         fields.lattice.evaluateStatistics();
427         fields.lattice.incrementTime();
428
429         lastIterationTime = global::timer("iteration").stop();
430     }
431 }
432
```

damBreak3d_4

```
/* This file is part of the Palabos library.
 *
 * Copyright (C) 2011-2015 FlowKit Sarl
 * Route d'Oron 2
 * 1010 Lausanne, Switzerland
 * E-mail contact: contact@flowkit.com
 *
 * The most recent release of Palabos can be downloaded at
 * <http://www.palabos.org/>
 *
 * The library Palabos is free software: you can redistribute it and/or
 * modify it under the terms of the GNU Affero General Public License as
 * published by the Free Software Foundation, either version 3 of the
 * License, or (at your option) any later version.
 *
 * The library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU Affero General Public License for more details.
 *
 * You should have received a copy of the GNU Affero General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The breaking dam free surface problem. This code demonstrates the basic usage of the
 * free surface module in Palabos. Surface tension and contact angles are optional.
 */

#include "palabos3D.h"
#include "palabos3D.hh"

using namespace plb;
using namespace std;

#define DESCRIPTOR descriptors::ForcedD3Q19Descriptor
typedef double T;


// Smagorinsky constant for LES model.
const T cSmago = 0.14;

// Physical dimensions of the system (in meters).
const T lx = 2.0;
const T ly = 0.3;
const T lz = 1.3;

const T rhoEmpty = T(1);

plint writeImagesIter   = 100;
plint getStatisticsIter = 20;

plint maxIter;
plint N;
plint nx, ny, nz;
T delta_t, delta_x;
Array<T,3> externalForce;
T nuPhys, nuLB, tau, omega, Bo, surfaceTensionLB, contactAngle;

std::string outDir;
plint obstacleCenterXYplane, obstacleLength, obstacleWidth, obstacleHeight, beginWaterReservoir,
waterReservoirHeight;
plint waterLevelOne, waterLevelTwo, waterLevelThree, waterLevelFour;

void setupParameters() {
    delta_x = lz / N;
    nx = util::roundToInt(lx / delta_x);
```

```
66      ny = util::roundToInt(ly / delta_x);
67      nz = util::roundToInt(lz / delta_x);
68
69      // Gravity in lattice units.
70      T gLB = 9.8 * delta_t * delta_t/delta_x;
71      externalForce = Array<T,3>(0., 0., -gLB);
72      tau            = (nuPhys*DESCRIPTOR<T>::invCs2*delta_t)/(delta_x*delta_x) + 0.5;
73      omega          = 1./tau;
74      nuLB           = (tau-0.5)*DESCRIPTOR<T>::cs2; // Viscosity in lattice units.
75
76      surfaceTensionLB = rhoEmpty * gLB * N * N / Bo;
77
78      obstacleCenterXYplane = util::roundToInt(0.744*N);
79      obstacleLength        = util::roundToInt(0.403*N);
80      obstacleWidth         = util::roundToInt(0.161*N);
81      obstacleHeight        = util::roundToInt(0.161*N);
82      beginWaterReservoir   = util::roundToInt((1.43/2.0)*nx);
83      waterReservoirHeight  = util::roundToInt(nz);
84
85      waterLevelOne   = util::roundToInt(0.496*N);
86      waterLevelTwo   = util::roundToInt(2.*0.496*N);
87      waterLevelThree = util::roundToInt(3.*0.496*N);
88      waterLevelFour  = util::roundToInt((3.*0.496 + 1.150)*N);
89  }
90
91  // Specifies the initial condition for the fluid (each cell is assigned the
92  // flag "fluid", "empty", or "wall").
93  int initialFluidFlags(plint iX, plint iY, plint iZ) {
94      // Place an obstacle on the left end, which is hit by the fluid.
95          bool insideObstacle =
96          iX >= (1.4/2.0)*nx &&
97          iX <= (1.43/2.0)*nx &&
98          iY >=  0 &&
99          iY <= ny &&
100         iZ >= 0 &&
101         iZ <= (0.58/1.3)*nz;
102
103 bool insideObstacle1 =
104         iX >= (1.1/2.0)*nx &&
105         iX <= (1.40/2.0)*nx &&
106         iY >=  0 &&
107         iY <= ny &&
108         iZ >= 0 &&
109         iZ <= (0.48/1.3)*nz;
110
111 bool insideObstacle2 =
112         iX >= (0.8/2.0)*nx &&
113         iX <= (1.1/2.0)*nx &&
114         iY >=  0 &&
115         iY <= ny &&
116         iZ >= 0 &&
117         iZ <= (0.36/1.3)*nz;
118
119 bool insideObstacle3 =
120         iX >= (0.5/2.0)*nx &&
121         iX <= (0.8/2.0)*nx &&
122         iY >=  0 &&
123         iY <=  ny &&
124         iZ >=  0 &&
125         iZ <= (0.24/1.3)*nz;
126
127 bool insideObstacle4 =
128         iX >= (0.2/2.0)*nx &&
129         iX <= (0.5/2.0)*nx &&
130         iY >=  0 &&
131         iY <=  ny &&
```

```
        iZ >=  0 &&
        iZ <= (0.12/1.3)*nz;

bool insideObstacle5 =
        iX >=  0 &&
        iX <= (0.2/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <= (0.03/1.3)*nz;

bool insideObstacle6 =
        iX >= (0.2/2.0)*nx &&
        iX <= (0.21/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >= (0.12/1.3)*nz &&
        iZ <= (0.142/1.3)*nz;

bool insideObstacle7 =
        iX >=  (0.5/2.0)*nx &&
        iX <=  (0.51/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.24/1.3)*nz &&
        iZ <=  (0.262/1.3)*nz;

bool insideObstacle8 =
        iX >=  (0.8/2.0)*nx &&
        iX <=  (0.81/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.36/1.3)*nz &&
        iZ <=  (0.382/1.3)*nz;

bool insideObstacle9 =
        iX >=  (1.1/2.0)*nx &&
        iX <=  (1.11/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.48/1.3)*nz &&
        iZ <=  (0.502/1.3)*nz;

bool insideObstacle10 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  0 &&
        iY <=  (0.1/0.3)*ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle11 =
        iX >=  0 &&
        iX <=  (1.4/2.0)*nx &&
        iY >=  (0.2/0.3)*ny &&
        iY <=  ny &&
        iZ >=  0 &&
        iZ <=  nz;

bool insideObstacle12 =
        iX >=  (1.8/2.0)*nx &&
        iX <=  (1.85/2.0)*nx &&
        iY >=  0 &&
        iY <=  ny &&
        iZ >=  (0.2/1.3)*nz &&
        iZ <=  nz;
```

```
198
199     if (insideObstacle) {
200         return twoPhaseFlag::wall;
201     }
202     else if (insideObstacle1) {
203         return twoPhaseFlag::wall;
204     }
205     else if (insideObstacle2) {
206         return twoPhaseFlag::wall;
207     }
208     else if (insideObstacle3) {
209         return twoPhaseFlag::wall;
210     }
211     else if (insideObstacle4) {
212         return twoPhaseFlag::wall;
213     }
214     else if (insideObstacle5) {
215         return twoPhaseFlag::wall;
216     }
217     else if (insideObstacle6) {
218         return twoPhaseFlag::wall;
219     }
220     else if (insideObstacle7) {
221         return twoPhaseFlag::wall;
222     }
223     else if (insideObstacle8) {
224         return twoPhaseFlag::wall;
225     }
226     else if (insideObstacle9) {
227         return twoPhaseFlag::wall;
228     }
229     else if (insideObstacle10) {
230         return twoPhaseFlag::wall;
231     }
232     else if (insideObstacle11) {
233         return twoPhaseFlag::wall;
234     }
235     else if (insideObstacle12) {
236         return twoPhaseFlag::wall;
237     }
238
239     else if (iX >= beginWaterReservoir && iZ <= waterReservoirHeight) {
240         return twoPhaseFlag::fluid;
241     }
242     else {
243         return twoPhaseFlag::empty;
244     }
245 }
246
247 void writeResults(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, MultiScalarField3D<T>& volumeFraction, plint
iT)
248 {
249     static const plint nx = lattice.getNx();
250     static const plint ny = lattice.getNy();
251     static const plint nz = lattice.getNz();
252     Box3D slice(0, nx-1, ny/2, ny/2, 0, nz-1);
253     ImageWriter<T> imageWriter("leeloo");
254     imageWriter.writeScaledPpm(createFileName("u", iT, 6),
255                                *computeVelocityNorm(lattice, slice));
256
257     imageWriter.writeScaledPpm(createFileName("rho", iT, 6),
258                                *computeDensity(lattice, slice));
259
260     imageWriter.writeScaledPpm(createFileName("volumeFraction", iT, 6), *extractSubDomain(volumeFraction,
slice));
261
```

```
262     // Use a marching-cube algorithm to reconstruct the free surface and write an STL file.
263     std::vector<T> isoLevels;
264     isoLevels.push_back((T) 0.5);
265     typedef TriangleSet<T>::Triangle Triangle;
266     std::vector<Triangle> triangles;
267     isoSurfaceMarchingCube(triangles, volumeFraction, isoLevels, volumeFraction.getBoundingBox());
268     TriangleSet<T>(triangles).writeBinarySTL(createFileName(outDir+"/interface", iT, 6)+".stl");
269
270     VtkImageOutput3D<T> vtkOut(createFileName("volumeFraction", iT, 6), 1.);
271     vtkOut.writeData<float>(volumeFraction, "vf", 1.);
272 }
273
274 //void writeStatistics(FreeSurfaceFields3D<T,DESCRIPTOR>& fields) {
275 void writeStatistics(TwoPhaseFields3D<T,DESCRIPTOR>& fields) {
276     pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
277     T averageMass = freeSurfaceAverageMass<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
278     pcout << "Average Mass: " << averageMass  << endl;
279     T averageDensity = freeSurfaceAverageDensity<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
280     pcout << "Average Density: " << setprecision(12) << averageDensity  << endl;
281
282     T averageVolumeFraction = freeSurfaceAverageVolumeFraction<T,DESCRIPTOR>(fields.twoPhaseArgs,
fields.lattice.getBoundingBox());
283     pcout << "Average Volume-Fraction: " << setprecision(12) << averageVolumeFraction  << endl;
284
285     pcout << " -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- " << endl;
286 }
287
288 void writeResultsP(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
289 {
290     static const plint nx = lattice.getNx();
291     static const plint ny = lattice.getNy();
292     static const plint nz = lattice.getNz();
293     Box3D boundingBoxP(0, nx-1, 0, ny-1, 0, nz-1);
294
295     VtkImageOutput3D<T>vtkOutP(createFileName("Pressure", iT, 8), 1.);
296     std::auto_ptr<MultiScalarField3D<T> > px = fields->computePressure(boundingBoxP);
297     vtkOutP.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
298
299 //     std::auto_ptr<MultiTensorField3D<T> > vx = fields->computeVelocity(boundingBoxP);
300 //     vtkOutPV.writeData<3,float>(*vx, "v", delta_x / delta_t);
301
302 }
303
304
305 void writeResultsV(MultiBlockLattice3D<T,DESCRIPTOR>& lattice, TwoPhaseFields3D<T,DESCRIPTOR> *fields,
plint iT)
306 {
307     static const plint nx = lattice.getNx();
308     static const plint ny = lattice.getNy();
309     static const plint nz = lattice.getNz();
310     Box3D boundingBoxV(0, nx-1, 0, ny-1, 0, nz-1);
311
312     VtkImageOutput3D<T>vtkOutV(createFileName("Velocity", iT, 8), 1.);
313 //     std::auto_ptr<MultiScalarField3D<T>> px = fields->computePressure(boundingBoxP);
314 //     vtkOutPV.writeData<float>(*px, "p", (delta_x * delta_x) / (delta_t * delta_t));
315
316     std::auto_ptr<MultiTensorField3D<T,3> > vx = fields->computeVelocity(boundingBoxV);
317     vtkOutV.writeData<3,float>(*vx, "v", delta_x / delta_t);
318
319 }
320
321 int main(int argc, char **argv)
322 {
```

```
323     plbInit(&argc, &argv);
324     global::directories().setInputDir("./");
325
326     if (global::argc() != 8) {
327         pcout << "Error missing some input parameter\n";
328     }
329
330     try {
331         global::argv(1).read(outDir);
332         global::directories().setOutputDir(outDir+"/");
333
334         global::argv(2).read(nuPhys);
335         global::argv(3).read(Bo);
336         global::argv(4).read(contactAngle);
337         global::argv(5).read(N);
338         global::argv(6).read(delta_t);
339         global::argv(7).read(maxIter);
340     }
341     catch(PlbIOException& except) {
342         pcout << except.what() << std::endl;
343         pcout << "The parameters for this program are :\n";
344         pcout << "1. Output directory name.\n";
345         pcout << "2. kinematic viscosity in physical Units (m^2/s) .\n";
346         pcout << "3. Bond number (Bo = rho * g * L^2 / gamma).\n";
347         pcout << "4. Contact angle (in degrees).\n";
348         pcout << "5. number of lattice nodes for lz .\n";
349         pcout << "6. delta_t .\n";
350         pcout << "7. maxIter .\n";
351         pcout << "Reasonable parameters on a desktop computer are: " << (std::string)global::argv(0) << "
tmp 1.e-5 100 80.0 40 1.e-3 80000\n";
352         pcout << "Reasonable parameters on a parallel machine are: " << (std::string)global::argv(0) << "
tmp 1.e-6 100 80.0 100 1.e-4 80000\n";
353         exit (EXIT_FAILURE);
354     }
355
356     setupParameters();
357
358     pcout << "delta_t= " << delta_t << endl;
359     pcout << "delta_x= " << delta_x << endl;
360     pcout << "delta_t*delta_t/delta_x= " << delta_t*delta_t/delta_x << endl;
361     pcout << "externalForce= " << externalForce[2] << endl;
362     pcout << "relaxation time= " << tau << endl;
363     pcout << "omega= " << omega << endl;
364     pcout << "kinematic viscosity physical units = " << nuPhys << endl;
365     pcout << "kinematic viscosity lattice units= " << nuLB << endl;
366
367     global::timer("initialization").start();
368
369
370     SparseBlockStructure3D blockStructure(createRegularDistribution3D(nx, ny, nz));
371
372     Dynamics<T,DESCRIPTOR>* dynamics
373         = new SmagorinskyBGKdynamics<T,DESCRIPTOR>(omega, cSmago);
374
375     // If surfaceTensionLB is 0, then the surface tension algorithm is deactivated.
376     // If contactAngle is less than 0, then the contact angle algorithm is deactivated.
377     TwoPhaseFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
378                                            surfaceTensionLB, contactAngle, externalForce );
379     //FreeSurfaceFields3D<T,DESCRIPTOR> fields( blockStructure, dynamics->clone(), rhoEmpty,
380     //                                          surfaceTensionLB, contactAngle, externalForce, false );
381     //integrateProcessingFunctional(new ShortenBounceBack3D<T,DESCRIPTOR>, fields.lattice.getBoundingBox(),
fields.twoPhaseArgs, 0);
382
383     // Set all outer-wall cells to "wall" (here, bulk-cells are also set to "wall", but it
384     // doesn't matter, because they are overwritten on the next line).
385     setToConstant(fields.flag, fields.flag.getBoundingBox(), (int)twoPhaseFlag::wall);
```

```
386     // In the bulk (all except outer wall layer), initialize the flags as specified by
387     // the function "initialFluidFlags".
388     setToFunction(fields.flag, fields.flag.getBoundingBox().enlarge(-1), initialFluidFlags);
389
390     fields.defaultInitialize();
391
392     pcout << "Time spent for setting up lattices: "
393           << global::timer("initialization").stop() << endl;
394     T lastIterationTime = T();
395
396     for (plint iT = 0; iT <= maxIter; ++iT) {
397         global::timer("iteration").restart();
398
399         T sum_of_mass_matrix = T();
400         T lost_mass = T();
401         if (iT % getStatisticsIter==0) {
402             pcout << endl;
403             pcout << "ITERATION = " << iT << endl;
404             pcout << "Time of last iteration is " << lastIterationTime << " seconds" << endl;
405             writeStatistics(fields);
406             sum_of_mass_matrix = fields.lattice.getInternalStatistics().getSum(0);
407             pcout << "Sum of mass matrix: " << sum_of_mass_matrix << std::endl;
408             lost_mass = fields.lattice.getInternalStatistics().getSum(1);
409             pcout << "Lost mass: " << lost_mass << std::endl;
410             pcout << "Total mass: " << sum_of_mass_matrix + lost_mass << std::endl;
411             pcout << "Interface cells: " << fields.lattice.getInternalStatistics().getIntSum(0) <<
std::endl;
412         }
413
414         if (iT % writeImagesIter == 0) {
415             global::timer("images").start();
416             writeResultsP(fields.lattice,&fields, iT);   //added for pressure results output
417             writeResultsV(fields.lattice,&fields, iT);   //added for velocity results output
418
419             writeResults(fields.lattice, fields.volumeFraction, iT);
420             pcout << "Total time spent for writing images: "
421                 << global::timer("images").stop() << endl;
422         }
423
424         // This includes the collision-streaming cycle, plus all free-surface operations.
425         fields.lattice.executeInternalProcessors();
426         fields.lattice.evaluateStatistics();
427         fields.lattice.incrementTime();
428
429         lastIterationTime = global::timer("iteration").stop();
430     }
431 }
432
```

Appendix G: Act2_Output-Fe2 and Mn

Act2_Output-Fe2

```
--Output from Act2 activity-activity diagram generator--

Temperature is 30.8 C;  Pressure is 1.013 bars

pH plotted on the X axis from 0 to 14
Eh (volts) (swapped for O2(aq)) plotted on the Y axis from -.75 to 1.25

Stability limits of water

 Reaction                                          Log K      Equation
-------------------------------------------------------------------------------------------
H2(g)  = 2*H+  + 2*e-                              -.1283     Y =-.06032  *X + .003871
O2(g)  + 4*H+  + 4*e-  = 2*H2O                     81.49      Y =-.06032  *X + 1.229
-------------------------------------------------------------------------------------------

Diagram for Fe++

 Basis species     Activity/Fugacity
-----------------------------------------------------------
 Fe++                  .11            (main species)
 H2O                   1              (solvent)
 H+                    -on X-axis-
 e-                    -on Y-axis-
-----------------------------------------------------------

Species and minerals in main system

                  Activity    Reaction                                                    Log K
-------------------------------------------------------------------------------------------------
 Fe++              .11        Fe++  = Fe++                                                 0.0000
 Fe+++             .11        Fe+++  + e-  = Fe++                                         12.9313
 Fe(OH)2           .11        Fe(OH)2  + 2*H+  = Fe++  + 2*H2O                            21.0162
 Fe(OH)2+          .11        Fe(OH)2+  + 2*H+  + e-  = Fe++  + 2*H2O                     18.3744
 Fe(OH)3           .11        Fe(OH)3  + 3*H+  + e-  = Fe++  + 3*H2O                      24.5790
 Fe(OH)3-          .11        Fe(OH)3-  + 3*H+  = Fe++  + 3*H2O                           33.6703
 Fe(OH)4-          .11        Fe(OH)4-  + 4*H+  + e-  = Fe++  + 4*H2O                     34.1733
 Fe2(OH)2++++      .055       Fe2(OH)2++++  + 2*H+  + 2*e-  = 2*Fe++  + 2*H2O             28.6320
 Fe3(OH)4(5+)      .03667     Fe3(OH)4(5+)  + 4*H+  + 3*e-  = 3*Fe++  + 4*H2O             44.9562
 FeOH+             .11        FeOH+  + H+  = Fe++  + H2O                                  10.0455
 FeOH++            .11        FeOH++  + H+  + e-  = Fe++  + H2O                           14.9811
 Fe(OH)2(ppd)      1          Fe(OH)2(ppd)  + 2*H+  = Fe++  + 2*H2O                       12.5554
 Fe(OH)3(ppd)      1          Fe(OH)3(ppd)  + 3*H+  + e-  = Fe++  + 3*H2O                 17.5658
 FeO(c)            1          FeO(c)  + 2*H+  = Fe++  + H2O                               11.0518
 Goethite          1          Goethite  + 3*H+  + e-  = Fe++  + 2*H2O                     13.2348
 Hematite          1          Hematite  + 6*H+  + 2*e-  = 2*Fe++  + 3*H2O                 25.4932
 Magnetite         1          Magnetite  + 8*H+  + 2*e-  = 3*Fe++  + 4*H2O                35.4586
 Wustite           1          Wustite  + 2*H+  + .106*e-  = .947*Fe++  + H2O               13.4134
-------------------------------------------------------------------------------------------------


   1     Y =   0.780                Fe++  = Fe+++  + e-
   2     X =  10.508                Fe++  + 2*H2O  = Fe(OH)2  + 2*H+
   3     Y =   1.108 -   0.121*X    Fe++  + 2*H2O  = Fe(OH)2+  + 2*H+  + e-
   4     Y =   1.483 -   0.181*X    Fe++  + 3*H2O  = Fe(OH)3  + 3*H+  + e-
   5     X =  11.223                Fe++  + 3*H2O  = Fe(OH)3-  + 3*H+
   6     Y =   2.061 -   0.241*X    Fe++  + 4*H2O  = Fe(OH)4-  + 4*H+  + e-
   7     Y =   0.883 -   0.060*X    2*Fe++  + 2*H2O  = Fe2(OH)2++++  + 2*H+  + 2*e-
   8     Y =   0.933 -   0.080*X    3*Fe++  + 4*H2O  = Fe3(OH)4(5+)  + 4*H+  + 3*e-
   9     X =  10.045                Fe++  + H2O  = FeOH+  + H+
  10     Y =   0.904 -   0.060*X    Fe++  + H2O  = FeOH++  + H+  + e-
  11     X =   6.757                Fe++  + 2*H2O  = Fe(OH)2(ppd)  + 2*H+
  12     Y =   1.117 -   0.181*X    Fe++  + 3*H2O  = Fe(OH)3(ppd)  + 3*H+  + e-
  13     X =   6.005                Fe++  + H2O  = FeO(c)  + 2*H+
  14     Y =   0.856 -   0.181*X    Fe++  + 2*H2O  = Goethite  + 3*H+  + e-
  15     Y =   0.827 -   0.181*X    2*Fe++  + 3*H2O  = Hematite  + 6*H+  + 2*e-
  16     Y =   1.156 -   0.241*X    3*Fe++  + 4*H2O  = Magnetite  + 8*H+  + 2*e-
  17     Y =   8.149 -   1.138*X    Fe++  + 1.056*H2O  = 1.056*Wustite  + 2.112*H+  + .1119*e-
  18     Y =  -0.488 +   0.121*X    Fe+++  + 2*H2O  + e-  = Fe(OH)2  + 2*H+
  19     X =   2.722                Fe+++  + 2*H2O  = Fe(OH)2+  + 2*H+
  20     X =   3.883                Fe+++  + 3*H2O  = Fe(OH)3  + 3*H+
  21     Y =  -1.251 +   0.181*X    Fe+++  + 3*H2O  + e-  = Fe(OH)3-  + 3*H+
  22     X =   5.310                Fe+++  + 4*H2O  = Fe(OH)4-  + 4*H+
  23     X =   1.713                2*Fe+++  + 2*H2O  = Fe2(OH)2++++  + 2*H+
  24     X =   1.901                3*Fe+++  + 4*H2O  = Fe3(OH)4(5+)  + 4*H+
  25     Y =   0.174 +   0.060*X    Fe+++  + H2O  + e-  = FeOH+  + H+
  26     X =   2.050                Fe+++  + H2O  = FeOH++  + H+
  27     Y =  -0.035 +   0.121*X    Fe+++  + 2*H2O  + e-  = Fe(OH)2(ppd)  + 2*H+
  28     X =   1.864                Fe+++  + 3*H2O  = Fe(OH)3(ppd)  + 3*H+
  29     Y =   0.056 +   0.121*X    Fe+++  + H2O  + e-  = FeO(c)  + 2*H+
  30     X =   0.421                Fe+++  + 2*H2O  = Goethite  + 3*H+
  31     X =   0.258                2*Fe+++  + 3*H2O  = Hematite  + 6*H+
  32     Y =   0.028 +   0.483*X    3*Fe+++  + 4*H2O  + e-  = Magnetite  + 8*H+
  33     Y =  -0.149 +   0.143*X    Fe+++  + 1.056*H2O  + .8881*e-  = 1.056*Wustite  + 2.112*H+
  34     Y =  -0.159                Fe(OH)2  = Fe(OH)2+  + e-
  35     Y =   0.215 -   0.060*X    Fe(OH)2  + H2O  = Fe(OH)3  + H+  + e-
  36     X =  12.654                Fe(OH)2  + H2O  = Fe(OH)3-  + H+
  37     Y =   0.794 -   0.121*X    Fe(OH)2  + 2*H2O  = Fe(OH)4-  + 2*H+  + e-
  38     Y =  -0.384 +   0.060*X    2*Fe(OH)2  + 2*H+  = Fe2(OH)2++++  + 2*H2O  + 2*e-
  39     Y =  -0.335 +   0.040*X    3*Fe(OH)2  + 2*H+  = Fe3(OH)4(5+)  + 2*H2O  + 3*e-
  40     X =  10.971                Fe(OH)2  + H+  = FeOH+  + H2O
  41     Y =  -0.364 +   0.060*X    Fe(OH)2  + H+  = FeOH++  + H2O  + e-
  42       Products  favored        Fe(OH)2  = Fe(OH)2(ppd)
  43     Y =  -0.150 -   0.060*X    Fe(OH)2  + H2O  = Fe(OH)3(ppd)  + H+  + e-
  44       Products  favored        Fe(OH)2  = FeO(c)  + H2O
  45     Y =  -0.412 -   0.060*X    Fe(OH)2  = Goethite  + H+  + e-
  46     Y =  -0.441 -   0.060*X    2*Fe(OH)2  = Hematite  + H2O  + 2*H+  + 2*e-
  47     Y =  -0.745 -   0.060*X    3*Fe(OH)2  = Magnetite  + 2*H2O  + 2*H+  + 2*e-
  48     Y =  -3.176 -   0.060*X    Fe(OH)2  = 1.056*Wustite  + .944*H2O  + .1119*H+  + .1119*e-
  49     X =   6.205                Fe(OH)2+  + H2O  = Fe(OH)3  + H+
  50     Y =  -0.923 +   0.060*X    Fe(OH)2+  + H2O  + e-  = Fe(OH)3-  + H+
  51     X =   7.899                Fe(OH)2+  + 2*H2O  = Fe(OH)4-  + 2*H+
  52     X =   3.730                2*Fe(OH)2+  + 2*H+  = Fe2(OH)2++++  + 2*H2O
  53     X =   4.363                3*Fe(OH)2+  + 2*H+  = Fe3(OH)4(5+)  + 2*H2O
  54     Y =   0.502 -   0.060*X    Fe(OH)2+  + H+  + e-  = FeOH+  + H2O
  55     X =   3.393                Fe(OH)2+  + H+  = FeOH++  + H2O
  56     Y =   0.293                Fe(OH)2+  + e-  = Fe(OH)2(ppd)
  57     X =   0.150                Fe(OH)2+  + H2O  = Fe(OH)3(ppd)  + H+
  58     Y =   0.384                Fe(OH)2+  + e-  = FeO(c)  + H2O
  59     X =  -4.181                Fe(OH)2+  = Goethite  + H+
  60     X =  -4.669                2*Fe(OH)2+  = Hematite  + H2O  + 2*H+
  61     Y =   1.013 +   0.121*X    3*Fe(OH)2+  + e-  = Magnetite  + 2*H2O  + 2*H+
  62     Y =   0.221 +   0.008*X    Fe(OH)2+  + .8881*e-  = 1.056*Wustite  + .944*H2O  + .1119*H+
  63     Y =  -0.548                Fe(OH)3  + e-  = Fe(OH)3-
  64     X =   9.594                Fe(OH)3  + H2O  = Fe(OH)4-  + H+
  65     X =   4.967                2*Fe(OH)3  + 4*H+  = Fe2(OH)2++++  + 4*H2O
  66     X =   5.468                3*Fe(OH)3  + 5*H+  = Fe3(OH)4(5+)  + 5*H2O
  67     Y =   0.877 -   0.121*X    Fe(OH)3  + 2*H+  + e-  = FeOH+  + 2*H2O
  68     X =   4.799                Fe(OH)3  + 2*H+  = FeOH++  + 2*H2O
  69     Y =   0.667 -   0.060*X    Fe(OH)3  + H+  + e-  = Fe(OH)2(ppd)  + H2O
```

```
139     Y =   0.393 -   0.060*X     Fe(OH)3(ppd)  + H+  + e-  = FeO(c)  + 2*H2O
140        Products  favored        Fe(OH)3(ppd)  = Goethite  + H2O
141        Products  favored        2*Fe(OH)3(ppd)  = Hematite  + 3*H2O
142     Y =   1.040 -   0.060*X     3*Fe(OH)3(ppd)  + H+  + e-  = Magnetite  + 5*H2O
143     Y =   0.231 -   0.060*X     Fe(OH)3(ppd)  + .8881*H+  + .8881*e-  = 1.056*Wustite  + 1.944*H2O
144     Y =   0.132 -   0.060*X     FeO(c)  + H2O  = Goethite  + H+  + e-
145     Y =   0.102 -   0.060*X     2*FeO(c)  + H2O  = Hematite  + 2*H+  + 2*e-
146     Y =   0.069 -   0.060*X     3*FeO(c)  + H2O  = Magnetite  + 2*H+  + 2*e-
147     Y =   1.677 -   0.060*X     FeO(c)  + .05597*H2O  = 1.056*Wustite  + .1119*H+  + .1119*e-
148        Products  favored        2*Goethite  = Hematite  + H2O
149     Y =   0.256 -   0.060*X     3*Goethite  + H+  + e-  = Magnetite  + 2*H2O
150     Y =  -0.063 -   0.060*X     Goethite  + .8881*H+  + .8881*e-  = 1.056*Wustite  + .944*H2O
151     Y =   0.168 -   0.060*X     1.5*Hematite  + H+  + e-  = Magnetite  + .5*H2O
152     Y =  -0.096 -   0.060*X     Hematite  + 1.776*H+  + 1.776*e-  = 2.112*Wustite  + .8881*H2O
153     Y =  -0.255 -   0.060*X     Magnetite  + 1.664*H+  + 1.664*e-  = 3.168*Wustite  + .8321*H2O
```

```
 70        Products  favored        Fe(OH)3  = Fe(OH)3(ppd)
 71     Y =   0.758 -   0.060*X     Fe(OH)3  + H+  + e-  = FeO(c)  + 2*H2O
 72        Products  favored        Fe(OH)3  = Goethite  + H2O
 73        Products  favored        2*Fe(OH)3  = Hematite  + 3*H2O
 74     Y =   2.135 -   0.060*X     3*Fe(OH)3  + H+  + e-  = Magnetite  + 5*H2O
 75     Y =   0.642 -   0.060*X     Fe(OH)3  + .8881*H+  + .8881*e-  = 1.056*Wustite  + 1.944*H2O
 76     Y =   0.030 -   0.060*X     Fe(OH)3-  + H2O  = Fe(OH)4-  + H+  + e-
 77     Y =  -1.148 +   0.121*X     2*Fe(OH)3-  + 4*H+  = Fe2(OH)2++++  + 4*H2O  + 2*e-
 78     Y =  -1.098 +   0.101*X     3*Fe(OH)3-  + 5*H+  = Fe3(OH)4(5+)  + 5*H2O  + 3*e-
 79     X =  11.812                 Fe(OH)3-  + 2*H+  = FeOH++  + 2*H2O
 80     Y =  -1.127 +   0.121*X     Fe(OH)3-  + 2*H+  = FeOH++  + 2*H2O  + e-
 81     X =  20.156                 Fe(OH)3-  + H+  = Fe(OH)2(ppd)  + H2O
 82     Y =  -0.914                 Fe(OH)3-  = Fe(OH)3(ppd)  + e-
 83     X =  21.660                 Fe(OH)3-  + H+  = FeO(c)  + 2*H2O
 84     Y =  -1.175                 Fe(OH)3-  = Goethite  + H2O  + e-
 85     Y =  -1.204                 2*Fe(OH)3-  = Hematite  + 3*H2O  + 2*e-
 86     Y =  -1.890 +   0.030*X     3*Fe(OH)3-  + H+  = Magnetite  + 5*H2O  + 2*e-
 87     Y =  -9.995 +   0.479*X     Fe(OH)3-  + .8881*H+  = 1.056*Wustite  + 1.944*H2O  + .1119*e-
 88     X =   6.510                 2*Fe(OH)4-  + 6*H+  = Fe2(OH)2++++  + 6*H2O
 89     X =   7.015                 3*Fe(OH)4-  + 8*H+  = Fe3(OH)4(5+)  + 8*H2O
 90     Y =   1.455 -   0.181*X     Fe(OH)4-  + 3*H+  + e-  = FeOH+  + 3*H2O
 91     X =   6.397                 Fe(OH)4-  + 3*H+  = FeOH++  + 3*H2O
 92     Y =   1.246 -   0.121*X     Fe(OH)4-  + 2*H+  + e-  = Fe(OH)2(ppd)  + 2*H2O
 93     X =  15.649                 Fe(OH)4-  + H+  = Fe(OH)3(ppd)  + H2O
 94     Y =   1.337 -   0.121*X     Fe(OH)4-  + 2*H+  + e-  = FeO(c)  + 3*H2O
 95     X =  19.980                 Fe(OH)4-  + H+  = Goethite  + 2*H2O
 96     X =  20.468                 2*Fe(OH)4-  + 2*H+  = Hematite  + 5*H2O
 97     Y =   3.871 -   0.241*X     3*Fe(OH)4-  + 4*H+  + e-  = Magnetite  + 8*H2O
 98     Y =   1.294 -   0.128*X     Fe(OH)4-  + 1.888*H+  + .8881*e-  = 1.056*Wustite  + 2.944*H2O
 99     X =   2.462                 1.5*Fe2(OH)2++++  + H2O  = Fe3(OH)4(5+)  + H+
100     Y =   0.277                 Fe2(OH)2++++  + 2*e-  = 2*FeOH+
101        Reactants favored        Fe2(OH)2++++  = 2*FeOH++
102     Y =   0.068 +   0.060*X     Fe2(OH)2++++  + 2*H2O  + 2*e-  = 2*Fe(OH)2(ppd)  + 2*H+
103     X =   1.940                 Fe2(OH)2++++  + 4*H2O  = 2*Fe(OH)3(ppd)  + 4*H+
104     Y =   0.159 +   0.060*X     Fe2(OH)2++++  + 2*e-  = 2*FeO(c)  + 2*H+
105     X =  -0.226                 Fe2(OH)2++++  + 2*H2O  = 2*Goethite  + 4*H+
106     X =  -0.470                 Fe2(OH)2++++  + H2O  = Hematite  + 4*H+
107     Y =   0.338 +   0.302*X     1.5*Fe2(OH)2++++  + H2O  + e-  = Magnetite  + 5*H+
108     Y =  -0.032 +   0.076*X     Fe2(OH)2++++  + .1119*H2O  + 1.776*e-  = 2.112*Wustite  + 2.224*H+
109     Y =   0.327 -   0.020*X     Fe3(OH)4(5+)  + H+  + 3*e-  = 3*FeOH+  + H2O
110     X =   1.453                 Fe3(OH)4(5+)  + H+  = 3*FeOH++  + H2O
111     Y =   0.118 +   0.040*X     Fe3(OH)4(5+)  + 2*H2O  + 3*e-  = 3*Fe(OH)2(ppd)  + 2*H+
112     X =   1.835                 Fe3(OH)4(5+)  + 5*H2O  = 3*Fe(OH)3(ppd)  + 5*H+
113     Y =   0.208 +   0.040*X     Fe3(OH)4(5+)  + 3*e-  = 3*FeO(c)  + H2O  + 2*H+
114     X =  -0.763                 Fe3(OH)4(5+)  + 2*H2O  = 3*Goethite  + 5*H+
115     X =  -1.056                 Fe3(OH)4(5+)  + .5*H2O  = 1.5*Hematite  + 5*H+
116     Y =   0.486 +   0.241*X     Fe3(OH)4(5+)  + e-  = Magnetite  + 4*H+
117     Y =   0.023 +   0.053*X     Fe3(OH)4(5+)  + 2.664*e-  = 3.168*Wustite  + .8321*H2O  + 2.336*H+
118     Y =   0.298                 FeOH+  = FeOH++  + e-
119     X =   3.469                 FeOH+  + H2O  = Fe(OH)2(ppd)  + H+
120     Y =   0.511 -   0.121*X     FeOH+  + 2*H2O  = Fe(OH)3(ppd)  + 2*H+  + e-
121     X =   1.965                 FeOH+  = FeO(c)  + H+
122     Y =   0.250 -   0.121*X     FeOH+  + H2O  = Goethite  + 2*H+  + e-
123     Y =   0.221 -   0.121*X     2*FeOH+  + H2O  = Hematite  + 4*H+  + 2*e-
124     Y =   0.247 -   0.151*X     3*FeOH+  + H2O  = Magnetite  + 5*H+  + 2*e-
125     Y =   2.736 -   0.599*X     FeOH+  + .05597*H2O  = 1.056*Wustite  + 1.112*H+  + .1119*e-
126     Y =   0.088 +   0.060*X     FeOH++  + H2O  + e-  = Fe(OH)2(ppd)  + H+
127     X =   1.772                 FeOH++  + 2*H2O  = Fe(OH)3(ppd)  + 2*H+
128     Y =   0.179 +   0.060*X     FeOH++  + e-  = FeO(c)  + H+
129     X =  -0.394                 FeOH++  + H2O  = Goethite  + 2*H+
130     X =  -0.638                 2*FeOH++  + H2O  = Hematite  + 4*H+
131     Y =   0.399 +   0.302*X     3*FeOH++  + H2O  + e-  = Magnetite  + 5*H+
132     Y =  -0.010 +   0.076*X     FeOH++  + .05597*H2O  + .8881*e-  = 1.056*Wustite  + 1.112*H+
133     Y =   0.302 -   0.060*X     Fe(OH)2(ppd)  + H2O  = Fe(OH)3(ppd)  + H+  + e-
134        Products  favored        Fe(OH)2(ppd)  = FeO(c)  + H2O
135     Y =   0.041 -   0.060*X     Fe(OH)2(ppd)  = Goethite  + H+  + e-
136     Y =   0.012 -   0.060*X     2*Fe(OH)2(ppd)  = Hematite  + H2O  + 2*H+  + 2*e-
137     Y =  -0.067 -   0.060*X     3*Fe(OH)2(ppd)  = Magnetite  + 2*H2O  + 2*H+  + 2*e-
138     Y =   0.867 -   0.060*X     Fe(OH)2(ppd)  = 1.056*Wustite  + .944*H2O  + .1119*H+  + .1119*e-
```

Main Diagram

	pH(1)	Eh (V)(1)	pH(2)	Eh (V)(2)	Equation	Type
Fe++	0.000	0.780	0.258	0.780	1	Upper
	0.258	0.780	5.462	-0.162	15	Upper
	5.462	-0.162	6.005	-0.293	16	Upper
	6.005	-0.293	6.005	-0.358	13	Right
Fe+++	0.258	0.780	0.000	0.780	1	Lower
	0.258	1.213	0.258	0.780	31	Right
FeO(c)	6.005	-0.293	13.586	-0.750	146	Upper
	6.005	-0.358	6.005	-0.293	13	Left
Hematite	14.000	-0.677	5.462	-0.162	151	Lower
	5.462	-0.162	0.258	0.780	15	Lower
	0.258	0.780	0.258	1.213	31	Left
Magnetite	5.462	-0.162	14.000	-0.677	151	Upper
	13.586	-0.750	6.005	-0.293	146	Lower
	6.005	-0.293	5.462	-0.162	16	Lower

Act2_Output-Fe21

```
--Output from Act2 activity-activity diagram generator--

Temperature is 30.8 C;  Pressure is 1.013 bars

pH plotted on the X axis from 0 to 14
Eh (volts) (swapped for O2(aq)) plotted on the Y axis from -10 to 0

Stability limits of water

 Reaction                                                Log K      Equation
-------------------------------------------------------------------------------------------------
 H2(g)  = 2*H+  + 2*e-                                   -.1283     Y =-1       *X + .06417
 O2(g)  + 4*H+  + 4*e-  = 2*H2O                          81.49      Y =-1       *X + 20.37
-------------------------------------------------------------------------------------------------

Diagram for Fe++

 Basis species     Activity/Fugacity
-------------------------------------------------------------
 Fe++                  1.259          (main species)
 H2O                   1              (solvent)
 H+                    -on X-axis-
 e-                    -on Y-axis-
-------------------------------------------------------------

Species and minerals in main system

                  Activity    Reaction                                                                  Log K
-------------------------------------------------------------------------------------------------------------
 Fe++              1.259      Fe++  = Fe++                                                             0.0000
 Fe+++             1.259      Fe+++  + e-  = Fe++                                                     12.9313
 Fe(OH)2           1.259      Fe(OH)2  + 2*H+  = Fe++  + 2*H2O                                        21.0162
 Fe(OH)2+          1.259      Fe(OH)2+  + 2*H+  + e-  = Fe++  + 2*H2O                                 18.3744
 Fe(OH)3           1.259      Fe(OH)3  + 3*H+  + e-  = Fe++  + 3*H2O                                  24.5790
 Fe(OH)3-          1.259      Fe(OH)3-  + 3*H+  = Fe++  + 3*H2O                                       33.6703
 Fe(OH)4-          1.259      Fe(OH)4-  + 4*H+  + e-  = Fe++  + 4*H2O                                 34.1733
 Fe2(OH)2++++      .6295      Fe2(OH)2++++  + 2*H+  + 2*e-  = 2*Fe++  + 2*H2O                         28.6320
 Fe3(OH)4(5+)      .4196      Fe3(OH)4(5+)  + 4*H+  + 3*e-  = 3*Fe++  + 4*H2O                         44.9562
 FeOH+             1.259      FeOH+  + H+  = Fe++  + H2O                                              10.0455
 FeOH++            1.259      FeOH++  + H+  + e-  = Fe++  + H2O                                       14.9811
 Fe(OH)2(ppd)      1          Fe(OH)2(ppd)  + 2*H+  = Fe++  + 2*H2O                                   12.5554
 Fe(OH)3(ppd)      1          Fe(OH)3(ppd)  + 3*H+  + e-  = Fe++  + 3*H2O                             17.5658
 FeO(c)            1          FeO(c)  + 2*H+  = Fe++  + H2O                                           11.0518
 Goethite          1          Goethite  + 3*H+  + e-  = Fe++  + 2*H2O                                 13.2348
 Hematite          1          Hematite  + 6*H+  + 2*e-  = 2*Fe++  + 3*H2O                             25.4932
 Magnetite         1          Magnetite  + 8*H+  + 2*e-  = 3*Fe++  + 4*H2O                           35.4586
 Wustite           1          Wustite  + 2*H+  + .106*e-  = .947*Fe++  + H2O                          13.4134
-------------------------------------------------------------------------------------------------------------


  No.     Line equation              Reaction
-------------------------------------------------------------------------------------------------------
    1     Y =  12.931                Fe++  = Fe+++  + e-
    2     X =  10.508                Fe++  + 2*H2O  = Fe(OH)2  + 2*H+
    3     Y =  18.374 -   2.000*X    Fe++  + 2*H2O  = Fe(OH)2+  + 2*H+  + e-
    4     Y =  24.579 -   3.000*X    Fe++  + 3*H2O  = Fe(OH)3  + 3*H+  + e-
    5     X =  11.223                Fe++  + 3*H2O  = Fe(OH)3-  + 3*H+
    6     Y =  34.173 -   4.000*X    Fe++  + 4*H2O  = Fe(OH)4-  + 4*H+  + e-
    7     Y =  14.115 -   1.000*X    2*Fe++  + 2*H2O  = Fe2(OH)2++++  + 2*H+  + 2*e-
    8     Y =  14.760 -   1.333*X    3*Fe++  + 4*H2O  = Fe3(OH)4(5+)  + 4*H+  + 3*e-
    9     X =  10.045                Fe++  + H2O  = FeOH+  + H+
   10     Y =  14.981 -   1.000*X    Fe++  + H2O  = FeOH++  + H+  + e-
   11     X =   6.228                Fe++  + 2*H2O  = Fe(OH)2(ppd)  + 2*H+
   12     Y =  17.466 -   3.000*X    Fe++  + 3*H2O  = Fe(OH)3(ppd)  + 3*H+  + e-
   13     X =   5.476                Fe++  + H2O  = FeO(c)  + 2*H+
   14     Y =  13.135 -   3.000*X    Fe++  + 2*H2O  = Goethite  + 3*H+  + e-
   15     Y =  12.647 -   3.000*X    2*Fe++  + 3*H2O  = Hematite  + 6*H+  + 2*e-
   16     Y =  17.579 -   4.000*X    3*Fe++  + 4*H2O  = Magnetite  + 8*H+  + 2*e-
   17     Y = 125.648 -  18.868*X    Fe++  + 1.056*H2O  = 1.056*Wustite  + 2.112*H+  + .1119*e-
   18     Y =  -8.085 +   2.000*X    Fe+++  + 2*H2O  + e-  = Fe(OH)2  + 2*H+
   19     X =   2.722                Fe+++  + 2*H2O  = Fe(OH)2+  + 2*H+
   20     X =   3.883                Fe+++  + 3*H2O  = Fe(OH)3  + 3*H+
   21     Y = -20.739 +   3.000*X    Fe+++  + 3*H2O  + e-  = Fe(OH)3-  + 3*H+
   22     X =   5.310                Fe+++  + 4*H2O  = Fe(OH)4-  + 4*H+
   23     X =   1.184                2*Fe+++  + 2*H2O  = Fe2(OH)2++++  + 2*H+
   24     X =   1.371                3*Fe+++  + 4*H2O  = Fe3(OH)4(5+)  + 4*H+
   25     Y =   2.886 +   1.000*X    Fe+++  + H2O  + e-  = FeOH+  + H+
   26     X =   2.050                Fe+++  + H2O  = FeOH++  + H+
   27     Y =   0.476 +   2.000*X    Fe+++  + 2*H2O  + e-  = Fe(OH)2(ppd)  + 2*H+
   28     X =   1.511                Fe+++  + 3*H2O  = Fe(OH)3(ppd)  + 3*H+
   29     Y =   1.980 +   2.000*X    Fe+++  + H2O  + e-  = FeO(c)  + 2*H+
   30     X =   0.068                Fe+++  + 2*H2O  = Goethite  + 3*H+
   31     X =  -0.095                2*Fe+++  + 3*H2O  = Hematite  + 6*H+
   32     Y =   3.635 +   8.000*X    3*Fe+++  + 4*H2O  + e-  = Magnetite  + 8*H+
   33     Y =  -1.276 +   2.378*X    Fe+++  + 1.056*H2O  + .8881*e-  = 1.056*Wustite  + 2.112*H+
   34     Y =  -2.642                Fe(OH)2  = Fe(OH)2+  + e-
   35     Y =   3.563 -   1.000*X    Fe(OH)2  + H2O  = Fe(OH)3  + H+  + e-
   36     X =  12.654                Fe(OH)2  + H2O  = Fe(OH)3-  + H+
   37     Y =  13.157 -   2.000*X    Fe(OH)2  + 2*H2O  = Fe(OH)4-  + 2*H+  + e-
   38     Y =  -6.901 +   1.000*X    2*Fe(OH)2  + 2*H+  = Fe2(OH)2++++  + 2*H2O  + 2*e-
   39     Y =  -6.257 +   0.667*X    3*Fe(OH)2  + 2*H+  = Fe3(OH)4(5+)  + 2*H2O  + 3*e-
   40     X =  10.971                Fe(OH)2  + H+  = FeOH+  + H2O
   41     Y =  -6.035 +   1.000*X    Fe(OH)2  + H+  = FeOH++  + H2O  + e-
   42        Products  favored       Fe(OH)2  = Fe(OH)2(ppd)
   43     Y =  -3.550 -   1.000*X    Fe(OH)2  + H2O  = Fe(OH)3(ppd)  + H+  + e-
   44        Products  favored       Fe(OH)2  = FeO(c)  + H2O
   45     Y =  -7.881 -   1.000*X    Fe(OH)2  = Goethite  + H+  + e-
   46     Y =  -8.370 -   1.000*X    2*Fe(OH)2  = Hematite  + H2O  + 2*H+  + 2*e-
   47     Y = -13.945 -   1.000*X    3*Fe(OH)2  = Magnetite  + 2*H2O  + 2*H+  + 2*e-
   48     Y = -62.110 -   1.000*X    Fe(OH)2  = 1.056*Wustite  + .944*H2O  + .1119*H+  + .1119*e-
   49     X =   6.205                Fe(OH)2+  + H2O  = Fe(OH)3  + H+
   50     Y = -15.296 +   1.000*X    Fe(OH)2+  + H2O  + e-  = Fe(OH)3-  + H+
```

```
 51     X =   7.899                 Fe(OH)2+  + 2*H2O  = Fe(OH)4-  + 2*H+
 52     X =   4.259                 2*Fe(OH)2+  + 2*H+  = Fe2(OH)2++++  + 2*H2O
 53     X =   5.422                 3*Fe(OH)2+  + 2*H+  = Fe3(OH)4(5+)  + 2*H2O
 54     Y =   8.329 -   1.000*X     Fe(OH)2+  + H+  + e-  = FeOH+  + H2O
 55     X =   3.393                 Fe(OH)2+  + H+  = FeOH++  + H2O
 56     Y =   5.919                 Fe(OH)2+  + e-  = Fe(OH)2(ppd)
 57     X =  -0.909                 Fe(OH)2+  + H2O  = Fe(OH)3(ppd)  + H+
 58     Y =   7.423                 Fe(OH)2+  + e-  = FeO(c)  + H2O
 59     X =  -5.240                 Fe(OH)2+  = Goethite  + H+
 60     X =  -5.728                 2*Fe(OH)2+  = Hematite  + H2O  + 2*H+
 61     Y =  19.965 +   2.000*X     3*Fe(OH)2+  + e-  = Magnetite  + 2*H2O  + 2*H+
 62     Y =   4.854 +   0.126*X     Fe(OH)2+  + .8881*e-  = 1.056*Wustite  + .944*H2O  + .1119*H+
 63     Y =  -9.091                 Fe(OH)3  + e-  = Fe(OH)3-
 64     X =   9.594                 Fe(OH)3  + H2O  = Fe(OH)4-  + H+
 65     X =   5.232                 2*Fe(OH)3  + 4*H+  = Fe2(OH)2++++  + 4*H2O
 66     X =   5.892                 3*Fe(OH)3  + 5*H+  = Fe3(OH)4(5+)  + 5*H2O
 67     Y =  14.534 -   2.000*X     Fe(OH)3  + 2*H+  + e-  = FeOH+  + 2*H2O
 68     X =   4.799                 Fe(OH)3  + 2*H+  = FeOH++  + 2*H2O
 69     Y =  12.124 -   1.000*X     Fe(OH)3  + H+  + e-  = Fe(OH)2(ppd)  + H2O
 70        Products  favored        Fe(OH)3  = Fe(OH)3(ppd)
 71     Y =  13.627 -   1.000*X     Fe(OH)3  + H+  + e-  = FeO(c)  + 2*H2O
 72        Products  favored        Fe(OH)3  = Goethite  + H2O
 73        Products  favored        2*Fe(OH)3  = Hematite  + 3*H2O
 74     Y =  38.578 -   1.000*X     3*Fe(OH)3  + H+  + e-  = Magnetite  + 5*H2O
 75     Y =  11.840 -   1.000*X     Fe(OH)3  + .8881*H+  + .8881*e-  = 1.056*Wustite  + 1.944*H2O
 76     Y =   0.503 -   1.000*X     Fe(OH)3-  + H2O  = Fe(OH)4-  + H+  + e-
 77     Y = -19.555 +   2.000*X     2*Fe(OH)3-  + 4*H+  = Fe2(OH)2++++  + 4*H2O  + 2*e-
 78     Y = -18.911 +   1.667*X     3*Fe(OH)3-  + 5*H+  = Fe3(OH)4(5+)  + 5*H2O  + 3*e-
 79     X =  11.812                 Fe(OH)3-  + 2*H+  = FeOH+  + 2*H2O
 80     Y = -18.689 +   2.000*X     Fe(OH)3-  + 2*H+  = FeOH++  + 2*H2O  + e-
 81     X =  21.215                 Fe(OH)3-  + H+  = Fe(OH)2(ppd)  + H2O
 82     Y = -16.205                 Fe(OH)3-  = Fe(OH)3(ppd)  + e-
 83     X =  22.719                 Fe(OH)3-  + H+  = FeO(c)  + 2*H2O
 84     Y = -20.536                 Fe(OH)3-  = Goethite  + H2O  + e-
 85     Y = -21.024                 2*Fe(OH)3-  = Hematite  + 3*H2O  + 2*e-
 86     Y = -32.926 +   0.500*X     3*Fe(OH)3-  + H+  = Magnetite  + 5*H2O  + 2*e-
 87     Y =-175.161 +   7.934*X     Fe(OH)3-  + .8881*H+  = 1.056*Wustite  + 1.944*H2O  + .1119*e-
 88     X =   6.686                 2*Fe(OH)4-  + 6*H+  = Fe2(OH)2++++  + 6*H2O
 89     X =   7.280                 3*Fe(OH)4-  + 8*H+  = Fe3(OH)4(5+)  + 8*H2O
 90     Y =  24.128 -   3.000*X     Fe(OH)4-  + 3*H+  + e-  = FeOH+  + 3*H2O
 91     X =   6.397                 Fe(OH)4-  + 3*H+  = FeOH++  + 3*H2O
 92     Y =  21.718 -   2.000*X     Fe(OH)4-  + 2*H+  + e-  = Fe(OH)2(ppd)  + 2*H2O
 93     X =  16.708                 Fe(OH)4-  + H+  = Fe(OH)3(ppd)  + H2O
 94     Y =  23.221 -   2.000*X     Fe(OH)4-  + 2*H+  + e-  = FeO(c)  + 3*H2O
 95     X =  21.039                 Fe(OH)4-  + H+  = Goethite  + 2*H2O
 96     X =  21.527                 2*Fe(OH)4-  + 2*H+  = Hematite  + 5*H2O
 97     Y =  67.361 -   4.000*X     3*Fe(OH)4-  + 4*H+  + e-  = Magnetite  + 8*H2O
 98     Y =  22.644 -   2.126*X     Fe(OH)4-  + 1.888*H+  + .8881*e-  = 1.056*Wustite  + 2.944*H2O
 99     X =   1.933                 1.5*Fe2(OH)2++++  + H2O  = Fe3(OH)4(5+)  + H+
100     Y =   4.070                 Fe2(OH)2++++  + 2*e-  = 2*FeOH+
101        Reactants favored        Fe2(OH)2++++  = 2*FeOH++
102     Y =   1.660 +   1.000*X     Fe2(OH)2++++  + 2*H2O  + 2*e-  = 2*Fe(OH)2(ppd)  + 2*H+
103     X =   1.675                 Fe2(OH)2++++  + 4*H2O  = 2*Fe(OH)3(ppd)  + 4*H+
104     Y =   3.164 +   1.000*X     Fe2(OH)2++++  + 2*e-  = 2*FeO(c)  + 2*H+
105     X =  -0.490                 Fe2(OH)2++++  + 2*H2O  = 2*Goethite  + 4*H+
106     X =  -0.734                 Fe2(OH)2++++  + H2O  = Hematite  + 4*H+
107     Y =   7.188 +   5.000*X     1.5*Fe2(OH)2++++  + H2O  + e-  = Magnetite  + 5*H+
108     Y =   0.058 +   1.252*X     Fe2(OH)2++++  + .1119*H2O  + 1.776*e-  = 2.112*Wustite  + 2.224*H+
109     Y =   4.714 -   0.333*X     Fe3(OH)4(5+)  + H+  + 3*e-  = 3*FeOH+  + H2O
110     X =  -0.664                 Fe3(OH)4(5+)  + H+  = 3*FeOH++  + H2O
111     Y =   2.304 +   0.667*X     Fe3(OH)4(5+)  + 2*H2O  + 3*e-  = 3*Fe(OH)2(ppd)  + 2*H+
112     X =   1.624                 Fe3(OH)4(5+)  + 5*H2O  = 3*Fe(OH)3(ppd)  + 5*H+
113     Y =   3.808 +   0.667*X     Fe3(OH)4(5+)  + 3*e-  = 3*FeO(c)  + H2O  + 2*H+
114     X =  -0.975                 Fe3(OH)4(5+)  + 2*H2O  = 3*Goethite  + 5*H+
115     X =  -1.268                 Fe3(OH)4(5+)  + .5*H2O  = 1.5*Hematite  + 5*H+
116     Y =   9.120 +   4.000*X     Fe3(OH)4(5+)  + e-  = Magnetite  + 4*H+
117     Y =   0.783 +   0.877*X     Fe3(OH)4(5+)  + 2.664*e-  = 3.168*Wustite  + .8321*H2O  + 2.336*H+
118     Y =   4.936                 FeOH+  = FeOH++  + e-
119     X =   2.410                 FeOH+  + H2O  = Fe(OH)2(ppd)  + H+


120     Y =   7.420 -   2.000*X     FeOH+  + 2*H2O  = Fe(OH)3(ppd)  + 2*H+  + e-
121     X =   0.906                 FeOH+  = FeO(c)  + H+
122     Y =   3.089 -   2.000*X     FeOH+  + H2O  = Goethite  + 2*H+  + e-
123     Y =   2.601 -   2.000*X     2*FeOH+  + H2O  = Hematite  + 4*H+  + 2*e-
124     Y =   2.511 -   2.500*X     3*FeOH+  + H2O  = Magnetite  + 5*H+  + 2*e-
125     Y =  35.902 -   9.934*X     FeOH+  + .05597*H2O  = 1.056*Wustite  + 1.112*H+  + .1119*e-
126     Y =   2.526 +   1.000*X     FeOH++  + H2O  + e-  = Fe(OH)2(ppd)  + H+
127     X =   1.242                 FeOH++  + 2*H2O  = Fe(OH)3(ppd)  + 2*H+
128     Y =   4.029 +   1.000*X     FeOH++  + e-  = FeO(c)  + H+
129     X =  -0.923                 FeOH++  + H2O  = Goethite  + 2*H+
130     X =  -1.167                 2*FeOH++  + H2O  = Hematite  + 4*H+
131     Y =   9.785 +   5.000*X     3*FeOH++  + H2O  + e-  = Magnetite  + 5*H+
132     Y =   1.033 +   1.252*X     FeOH++  + .05597*H2O  + .8881*e-  = 1.056*Wustite  + 1.112*H+
133     Y =   5.010 -   1.000*X     Fe(OH)2(ppd)  + H2O  = Fe(OH)3(ppd)  + H+  + e-
134        Products  favored        Fe(OH)2(ppd)  = FeO(c)  + H2O
135     Y =   0.679 -   1.000*X     Fe(OH)2(ppd)  = Goethite  + H+  + e-
136     Y =   0.191 -   1.000*X     2*Fe(OH)2(ppd)  = Hematite  + H2O  + 2*H+  + 2*e-
137     Y =  -1.104 -   1.000*X     3*Fe(OH)2(ppd)  = Magnetite  + 2*H2O  + 2*H+  + 2*e-
138     Y =  14.372 -   1.000*X     Fe(OH)2(ppd)  = 1.056*Wustite  + .944*H2O  + .1119*H+  + .1119*e-
139     Y =   6.514 -   1.000*X     Fe(OH)3(ppd)  + H+  + e-  = FeO(c)  + 2*H2O
140        Products  favored        Fe(OH)3(ppd)  = Goethite  + H2O
141        Products  favored        2*Fe(OH)3(ppd)  = Hematite  + 3*H2O
142     Y =  17.239 -   1.000*X     3*Fe(OH)3(ppd)  + H+  + e-  = Magnetite  + 5*H2O
143     Y =   3.830 -   1.000*X     Fe(OH)3(ppd)  + .8881*H+  + .8881*e-  = 1.056*Wustite  + 1.944*H2O
144     Y =   2.183 -   1.000*X     FeO(c)  + H2O  = Goethite  + H+  + e-
145     Y =   1.695 -   1.000*X     2*FeO(c)  + H2O  = Hematite  + 2*H+  + 2*e-
146     Y =   1.152 -   1.000*X     3*FeO(c)  + H2O  = Magnetite  + 2*H+  + 2*e-
147     Y =  27.805 -   1.000*X     FeO(c)  + .05597*H2O  = 1.056*Wustite  + .1119*H+  + .1119*e-
148        Products  favored        2*Goethite  = Hematite  + H2O
149     Y =   4.246 -   1.000*X     3*Goethite  + H+  + e-  = Magnetite  + 2*H2O
150     Y =  -1.046 -   1.000*X     Goethite  + .8881*H+  + .8881*e-  = 1.056*Wustite  + .944*H2O
151     Y =   2.781 -   1.000*X     1.5*Hematite  + H+  + e-  = Magnetite  + .5*H2O
152     Y =  -1.596 -   1.000*X     Hematite  + 1.776*H+  + 1.776*e-  = 2.112*Wustite  + .8881*H2O
153     Y =  -4.226 -   1.000*X     Magnetite  + 1.664*H+  + 1.664*e-  = 3.168*Wustite  + .8321*H2O
```

Main Diagram

	pH(1)	pe(1)	pH(2)	pe(2)	Equation	Type
Fe++	4.216	0.000	4.933	-2.152	15	Upper
	4.933	-2.152	5.476	-4.324	16	Upper
	5.476	-4.324	5.476	-5.412	13	Right
FeO(c)	5.476	-4.324	11.152	-10.000	146	Upper
	5.476	-5.412	5.476	-4.324	13	Left
Hematite	12.781	-10.000	4.933	-2.152	151	Lower
	4.933	-2.152	4.216	0.000	15	Lower
Magnetite	4.933	-2.152	12.781	-10.000	151	Upper
	11.152	-10.000	5.476	-4.324	146	Lower
	5.476	-4.324	4.933	-2.152	16	Lower

Act2_Output-Mn

--Output from Act2 activity-activity diagram generator--

Temperature is 30.8 C; Pressure is 1.013 bars

pH plotted on the X axis from 0 to 14
Eh (volts) (swapped for O2(aq)) plotted on the Y axis from -.75 to 1.25

Stability limits of water

Reaction	Log K	Equation
H2(g) = 2*H+ + 2*e-	-.1283	Y =-.06032 *X + .003871
O2(g) + 4*H+ + 4*e- = 2*H2O	81.49	Y =-.06032 *X + 1.229

Diagram for Mn++

Basis species	Activity/Fugacity	
Mn++	.008	(main species)
H2O	1	(solvent)
H+	-on X-axis-	
e-	-on Y-axis-	

Species and minerals in main system

	Activity	Reaction	Log K
Mn++	.008	Mn++ = Mn++	0.0000
MnO4-	.008	MnO4- + 8*H+ + 5*e- = Mn++ + 4*H2O	125.4246
MnO4--	.008	MnO4-- + 8*H+ + 4*e- = Mn++ + 4*H2O	116.3562
MnOH+	.008	MnOH+ + H+ = Mn++ + H2O	10.4040
Bixbyite	1	Bixbyite + 6*H+ + 2*e- = 2*Mn++ + 3*H2O	49.2064
Hausmannite	1	Hausmannite + 8*H+ + 2*e- = 3*Mn++ + 4*H2O	60.0738
Manganosite	1	Manganosite + 2*H+ = Mn++ + H2O	17.5418
Mn(OH)2(am)	1	Mn(OH)2(am) + 2*H+ = Mn++ + 2*H2O	14.9759
Mn(OH)3(c)	1	Mn(OH)3(c) + 3*H+ + e- = Mn++ + 3*H2O	31.3323
Pyrolusite	1	Pyrolusite + 4*H+ + 2*e- = Mn++ + 2*H2O	40.7979

No.	Line equation	Reaction
1	Y = 1.513 - 0.097*X	Mn++ + 4*H2O = MnO4- + 8*H+ + 5*e-
2	Y = 1.755 - 0.121*X	Mn++ + 4*H2O = MnO4-- + 8*H+ + 4*e-
3	X = 10.404	Mn++ + H2O = MnOH+ + H+
4	Y = 1.610 - 0.181*X	2*Mn++ + 3*H2O = Bixbyite + 6*H+ + 2*e-
5	Y = 2.001 - 0.241*X	3*Mn++ + 4*H2O = Hausmannite + 8*H+ + 2*e-
6	X = 9.819	Mn++ + H2O = Manganosite + 2*H+
7	X = 8.536	Mn++ + 2*H2O = Mn(OH)2(am) + 2*H+
8	Y = 2.016 - 0.181*X	Mn++ + 3*H2O = Mn(OH)3(c) + 3*H+ + e-
9	Y = 1.294 - 0.121*X	Mn++ + 2*H2O = Pyrolusite + 4*H+ + 2*e-
10	Y = 0.547	MnO4- + e- = MnO4--
11	Y = 1.388 - 0.084*X	MnO4- + 7*H+ + 5*e- = MnOH+ + 3*H2O
12	Y = 1.489 - 0.075*X	2*MnO4- + 10*H+ + 8*e- = Bixbyite + 5*H2O
13	Y = 1.438 - 0.074*X	3*MnO4- + 16*H+ + 13*e- = Hausmannite + 8*H2O
14	Y = 1.276 - 0.072*X	MnO4- + 6*H+ + 5*e- = Manganosite + 3*H2O
15	Y = 1.307 - 0.072*X	MnO4- + 6*H+ + 5*e- = Mn(OH)2(am) + 2*H2O
16	Y = 1.387 - 0.075*X	MnO4- + 5*H+ + 4*e- = Mn(OH)3(c) + H2O
17	Y = 1.659 - 0.080*X	MnO4- + 4*H+ + 3*e- = Pyrolusite + 2*H2O
18	Y = 1.598 - 0.106*X	MnO4-- + 7*H+ + 4*e- = MnOH+ + 3*H2O
19	Y = 1.803 - 0.101*X	2*MnO4-- + 10*H+ + 6*e- = Bixbyite + 5*H2O
20	Y = 1.705 - 0.097*X	3*MnO4-- + 16*H+ + 10*e- = Hausmannite + 8*H2O
21	Y = 1.458 - 0.090*X	MnO4-- + 6*H+ + 4*e- = Manganosite + 3*H2O
22	Y = 1.497 - 0.090*X	MnO4-- + 6*H+ + 4*e- = Mn(OH)2(am) + 2*H2O
23	Y = 1.667 - 0.101*X	MnO4-- + 5*H+ + 3*e- = Mn(OH)3(c) + H2O
24	Y = 2.215 - 0.121*X	MnO4-- + 4*H+ + 2*e- = Pyrolusite + 2*H2O
25	Y = 0.983 - 0.121*X	2*MnOH+ + H2O = Bixbyite + 4*H+ + 2*e-
26	Y = 1.060 - 0.151*X	3*MnOH+ + H2O = Hausmannite + 5*H+ + 2*e-
27	X = 9.235	MnOH+ = Manganosite + H+
28	X = 6.669	MnOH+ + H2O = Mn(OH)2(am) + H+
29	Y = 1.389 - 0.121*X	MnOH+ + 2*H2O = Mn(OH)3(c) + 2*H+ + e-
30	Y = 0.980 - 0.090*X	MnOH+ + H2O = Pyrolusite + 3*H+ + 2*e-
31	Y = 0.828 - 0.060*X	1.5*Bixbyite + H+ + e- = Hausmannite + .5*H2O
32	Y = 0.426 - 0.060*X	Bixbyite + 2*H+ + 2*e- = 2*Manganosite + H2O
33	Y = 0.581 - 0.060*X	Bixbyite + H2O + 2*H+ + 2*e- = 2*Mn(OH)2(am)
34	Reactants favored	Bixbyite + 3*H2O = 2*Mn(OH)3(c)
35	Y = 0.977 - 0.060*X	Bixbyite + H2O = 2*Pyrolusite + 2*H+ + 2*e-
36	Y = 0.225 - 0.060*X	Hausmannite + 2*H+ + 2*e- = 3*Manganosite + H2O
37	Y = 0.457 - 0.060*X	Hausmannite + 2*H2O + 2*H+ + 2*e- = 3*Mn(OH)2(am)
38	Y = 2.046 - 0.060*X	Hausmannite + 5*H2O = 3*Mn(OH)3(c) + H+ + e-
39	Y = 0.940 - 0.060*X	Hausmannite + 2*H2O = 3*Pyrolusite + 4*H+ + 4*e-
40	Products favored	Manganosite + H2O = Mn(OH)2(am)
41	Y = 0.832 - 0.060*X	Manganosite + 2*H2O = Mn(OH)3(c) + H+ + e-
42	Y = 0.701 - 0.060*X	Manganosite + H2O = Pyrolusite + 2*H+ + 2*e-
43	Y = 0.987 - 0.060*X	Mn(OH)2(am) + H2O = Mn(OH)3(c) + H+ + e-
44	Y = 0.779 - 0.060*X	Mn(OH)2(am) = Pyrolusite + 2*H+ + 2*e-
45	Y = 0.571 - 0.060*X	Mn(OH)3(c) = Pyrolusite + H2O + H+ + e-

Main Diagram

	pH(1)	Eh (V)(1)	pH(2)	Eh (V)(2)	Equation	Type
Mn++	1.076	1.164	5.253	0.660	9	Upper
	5.253	0.660	6.482	0.438	4	Upper
	6.482	0.438	8.536	-0.058	5	Upper
	8.536	-0.058	8.536	-0.511	7	Right
Bixbyite	5.253	0.660	14.000	0.132	35	Upper
	14.000	-0.016	6.482	0.438	31	Lower
	6.482	0.438	5.253	0.660	4	Lower
Hausmannite	6.482	0.438	14.000	-0.016	31	Upper
	14.000	-0.388	8.536	-0.058	37	Lower
	8.536	-0.058	6.482	0.438	5	Lower
Mn(OH)2(am)	8.536	-0.058	14.000	-0.388	37	Upper
	8.536	-0.511	8.536	-0.058	7	Left
Pyrolusite	14.000	0.132	5.253	0.660	35	Lower
	5.253	0.660	1.076	1.164	9	Lower

Act2_Output-Mn-1

--Output from Act2 activity-activity diagram generator--

Temperature is 30.8 C; Pressure is 1.013 bars

pH plotted on the X axis from 0 to 14
Eh (volts) (swapped for O2(aq)) plotted on the Y axis from -.75 to 1.25

Stability limits of water

Reaction	Log K	Equation
H2(g) = 2*H+ + 2*e-	-.1283	Y =-.06032 *X + .003871
O2(g) + 4*H+ + 4*e- = 2*H2O	81.49	Y =-.06032 *X + 1.229

Diagram for Mn++

Basis species	Activity/Fugacity	
Mn++	1.019	(main species)
H2O	1	(solvent)
H+	-on X-axis-	
e-	-on Y-axis-	

Species and minerals in main system

	Activity	Reaction	Log K
Mn++	1.019	Mn++ = Mn++	0.0000
MnO4-	1.019	MnO4- + 8*H+ + 5*e- = Mn++ + 4*H2O	125.4246
MnO4--	1.019	MnO4-- + 8*H+ + 4*e- = Mn++ + 4*H2O	116.3562
MnOH+	1.019	MnOH+ + H+ = Mn++ + H2O	10.4040
Bixbyite	1	Bixbyite + 6*H+ + 2*e- = 2*Mn++ + 3*H2O	49.2064
Hausmannite	1	Hausmannite + 8*H+ + 2*e- = 3*Mn++ + 4*H2O	60.0738
Manganosite	1	Manganosite + 2*H+ = Mn++ + H2O	17.5418
Mn(OH)2(am)	1	Mn(OH)2(am) + 2*H+ = Mn++ + 2*H2O	14.9759
Mn(OH)3(c)	1	Mn(OH)3(c) + 3*H+ + e- = Mn++ + 3*H2O	31.3323
Pyrolusite	1	Pyrolusite + 4*H+ + 2*e- = Mn++ + 2*H2O	40.7979

No.	Line equation	Reaction
1	Y = 1.513 - 0.097*X	Mn++ + 4*H2O = MnO4- + 8*H+ + 5*e-
2	Y = 1.755 - 0.121*X	Mn++ + 4*H2O = MnO4-- + 8*H+ + 4*e-
3	X = 10.404	Mn++ + H2O = MnOH+ + H+
4	Y = 1.483 - 0.181*X	2*Mn++ + 3*H2O = Bixbyite + 6*H+ + 2*e-
5	Y = 1.811 - 0.241*X	3*Mn++ + 4*H2O = Hausmannite + 8*H+ + 2*e-
6	X = 8.767	Mn++ + H2O = Manganosite + 2*H+
7	X = 7.484	Mn++ + 2*H2O = Mn(OH)2(am) + 2*H+
8	Y = 1.889 - 0.181*X	Mn++ + 3*H2O = Mn(OH)3(c) + 3*H+ + e-
9	Y = 1.230 - 0.121*X	Mn++ + 2*H2O = Pyrolusite + 4*H+ + 2*e-
10	Y = 0.547	MnO4- + e- = MnO4--
11	Y = 1.388 - 0.084*X	MnO4- + 7*H+ + 5*e- = MnOH+ + 3*H2O
12	Y = 1.520 - 0.075*X	2*MnO4- + 10*H+ + 8*e- = Bixbyite + 5*H2O
13	Y = 1.467 - 0.074*X	3*MnO4- + 16*H+ + 13*e- = Hausmannite + 8*H2O
14	Y = 1.302 - 0.072*X	MnO4- + 6*H+ + 5*e- = Manganosite + 3*H2O
15	Y = 1.332 - 0.072*X	MnO4- + 6*H+ + 5*e- = Mn(OH)2(am) + 2*H2O
16	Y = 1.419 - 0.075*X	MnO4- + 5*H+ + 4*e- = Mn(OH)3(c) + H2O
17	Y = 1.702 - 0.080*X	MnO4- + 4*H+ + 3*e- = Pyrolusite + 2*H2O
18	Y = 1.598 - 0.106*X	MnO4-- + 7*H+ + 4*e- = MnOH+ + 3*H2O
19	Y = 1.845 - 0.101*X	2*MnO4-- + 10*H+ + 6*e- = Bixbyite + 5*H2O
20	Y = 1.743 - 0.097*X	3*MnO4-- + 16*H+ + 10*e- = Hausmannite + 8*H2O
21	Y = 1.490 - 0.090*X	MnO4-- + 6*H+ + 4*e- = Manganosite + 3*H2O
22	Y = 1.529 - 0.090*X	MnO4-- + 6*H+ + 4*e- = Mn(OH)2(am) + 2*H2O
23	Y = 1.710 - 0.101*X	MnO4-- + 5*H+ + 3*e- = Mn(OH)3(c) + H2O
24	Y = 2.279 - 0.121*X	MnO4-- + 4*H+ + 2*e- = Pyrolusite + 2*H2O
25	Y = 0.856 - 0.121*X	2*MnOH+ + H2O = Bixbyite + 4*H+ + 2*e-
26	Y = 0.870 - 0.151*X	3*MnOH+ + H2O = Hausmannite + 5*H+ + 2*e-
27	X = 7.130	MnOH+ = Manganosite + H+
28	X = 4.564	MnOH+ + H2O = Mn(OH)2(am) + H+
29	Y = 1.262 - 0.121*X	MnOH+ + 2*H2O = Mn(OH)3(c) + 2*H+ + e-
30	Y = 0.916 - 0.090*X	MnOH+ + H2O = Pyrolusite + 3*H+ + 2*e-
31	Y = 0.828 - 0.060*X	1.5*Bixbyite + H+ + e- = Hausmannite + .5*H2O
32	Y = 0.426 - 0.060*X	Bixbyite + 2*H+ + 2*e- = 2*Manganosite + H2O
33	Y = 0.581 - 0.060*X	Bixbyite + H2O + 2*H+ + 2*e- = 2*Mn(OH)2(am)
34	Reactants favored	Bixbyite + 3*H2O = 2*Mn(OH)3(c)
35	Y = 0.977 - 0.060*X	Bixbyite + H2O = 2*Pyrolusite + 2*H+ + 2*e-
36	Y = 0.225 - 0.060*X	Hausmannite + 2*H+ + 2*e- = 3*Manganosite + H2O
37	Y = 0.457 - 0.060*X	Hausmannite + 2*H2O + 2*H+ + 2*e- = 3*Mn(OH)2(am)
38	Y = 2.046 - 0.060*X	Hausmannite + 5*H2O = 3*Mn(OH)3(c) + H+ + e-
39	Y = 0.940 - 0.060*X	Hausmannite + 2*H2O = 3*Pyrolusite + 4*H+ + 4*e-
40	Products favored	Manganosite + H2O = Mn(OH)2(am)
41	Y = 0.832 - 0.060*X	Manganosite + 2*H2O = Mn(OH)3(c) + H+ + e-
42	Y = 0.701 - 0.060*X	Manganosite + H2O = Pyrolusite + 2*H+ + 2*e-
43	Y = 0.987 - 0.060*X	Mn(OH)2(am) + H2O = Mn(OH)3(c) + H+ + e-
44	Y = 0.779 - 0.060*X	Mn(OH)2(am) = Pyrolusite + 2*H+ + 2*e-
45	Y = 0.571 - 0.060*X	Mn(OH)3(c) = Pyrolusite + H2O + H+ + e-

Main Diagram

	pH(1)	Eh (V)(1)	pH(2)	Eh (V)(2)	Equation	Type
Mn++	0.023	1.227	4.200	0.723	9	Upper
	4.200	0.723	5.430	0.501	4	Upper
	5.430	0.501	7.484	0.005	5	Upper
	7.484	0.005	7.484	-0.448	7	Right
Bixbyite	4.200	0.723	14.000	0.132	35	Upper
	14.000	-0.016	5.430	0.501	31	Lower
	5.430	0.501	4.200	0.723	4	Lower
Hausmannite	5.430	0.501	14.000	-0.016	31	Upper
	14.000	-0.388	7.484	0.005	37	Lower
	7.484	0.005	5.430	0.501	5	Lower
Mn(OH)2(am)	7.484	0.005	14.000	-0.388	37	Upper
	7.484	-0.448	7.484	0.005	7	Left
Pyrolusite	14.000	0.132	4.200	0.723	35	Lower
	4.200	0.723	0.023	1.227	9	Lower

Appendix H: Groundwater Characteristic

CHARACTERISTIC	1	2	3	4	5	6	7	8	9	10	11	12
TEMPERATURE, ^{O}C	30.8	30.51	30.9	30.8	30.8	30.7	30.95	30.91	30.9	30.81	30.9	31.2
pH	6.12	6.13	6.15	6.14	6.14	6.15	6.14	6.13	6.14	6.14	6.12	6.14
COLOUR PtCo (465nm)	-1	-1	-2	-2	-3	-1	-2	-2	-2	-1	-3	-4
TURBIDITY	10.54	6.13	10.12	12.1	14.76	7.51	11.6	13.1	6.12	10.53	11.23	11.53
Ntu												
CHEMICAL OXYGEN DEMAND [COD] mg/L	8	5.5	5.5	5.1	5.6	6.2	6.6	6.2	5.1	4.4	4	4.3
BIOCHEMICAL OXYGEN DEMAND	2.1	2.3	1.7	1.7	1.8	2.4	1.9	2.3	2.4	1.62	1.7	2.2
[BOD] mg/L												
TOTAL DISSOLVED SOLIDS [TDS] mg/L	132.21	140.1	141.2	131.41	144.62	130	136.55	142.33	143.11	147.55	144.21	132.1
SALINITY (ppt)	0.1	0.11	0.09	0.89	0.11	0.1	0.009	0.1	0.11	0.11	0.1	0.1
DISSOLVED OXYGEN ON SITE mg/L	1.5	1.5	1.55	1.56	1.52	1.6	1.51	1.54	1.56	1.55	1.57	1.55
CONDUCTIVITY uS/cm	233.56	235.16	222.91	222.8	244.12	234.15	246.5	245.91	234.45	239.1	245.3	234.66
Ferum (Fe) mg/L	11.1	11.4	11	11.2	11.1	11.1	11.3	11.5	11.3	11.2	11.1	11.4
Manganese (Mn) mg/L	0.55	0.52	0.5	0.53	0.51	0.53	0.54	0.53	0.56	0.52	0.53	0.54

Appendix I: WHO Guideline

Table 2 - Other Water Quality Parameters

Parameter	Existing Standard	Parameter	Standard for the Reprovisioned Sha Tin WTW South Works
pH at 25°C	8.2 – 8.8	pH at 25°C	8.2 – 8.8
Colour	Not exceeding 5 Hazen units	Colour	Not exceeding 5 Hazen units
Turbidity	Not exceeding 1.5 NTU	Turbidity	Not exceeding 1.0 NTU, and not exceeding 0.3 NTU in 95% of daily samples in any month
Iron as Fe	Not exceeding 0.1 mg/L	Iron as Fe	Not exceeding 0.1 mg/L
Manganese as Mn	Not exceeding 0.05 mg/L	Manganese as Mn	Not exceeding 0.05 mg/L
Aluminium as Al	Not exceeding 0.10 mg/L	Aluminium as Al	Not exceeding 0.10 mg/L
Free residual chlorine	0.5 - 1.5 mg/L	Free residual chlorine	0.5 - 1.5 mg/L
Fluoride as F	± 10% of norminal level (current 0.5 mg/L)	Fluoride as F	± 10% of norminal level (current 0.5 mg/L)
Taste and odour	Unobjectionable	Taste and odour	Unobjectionable
Total Coliforms & E.coli (no./100mL)	Absent	Total Coliforms & E.coli (no./100mL)	Absent
-	-	Cryptosporidium	4-log (99.99%) reduction or inactivation
-	-	Giardia	4-log (99.99%) reduction or inactivation
-	-	Viruses	4-log (99.99%) reduction or inactivation

Appendix J: MOH Guideline

Drinking Water Quality Standard

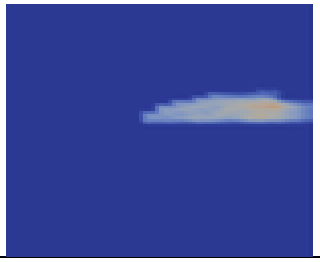
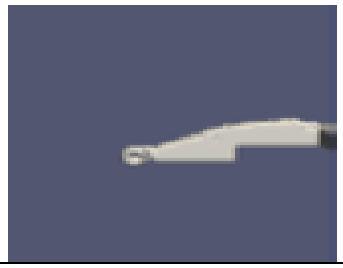
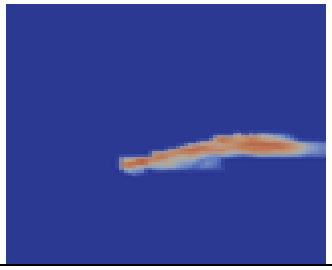
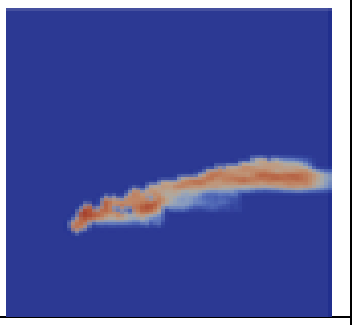
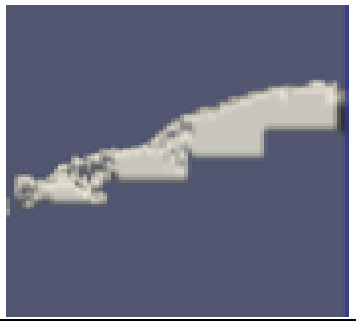
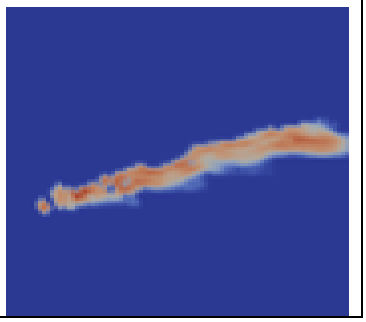
Engineering Services Division, Ministry of Health Malaysia

Parameter	Group	RECOMMENDED RAW WATER QUALITY	DRINKING WATER QUALITY STANDARDS
		Acceptable Value (mg/litre (unless otherwise stated))	Maximum Acceptable Value (mg/litre (unless otherwise stated))
Total Coliform	1	5000 MPN / 100 ml	0 in 100 ml
E.coli	1	5000 MPN / 100 m	0 in 100 m
Turbidity	1	1000 NTU	5 NTU
Color	1	300 TCU	15 TCU
pH	1	5.5 - 9.0	6.5 - 9.0
Free Residual Chlorine	1	-	0.2 - 5.0
Combined Chlorine	1	-	Not Less Than 1.0
Temperature	1	-	-
Clostridium perfringens (including spores)	1	-	Absent
Coliform bacteria	1	-	-
Colony count 22°	1	-	-
Conductivity	1	-	-
Enterococci	1	-	-
Odour	1	-	-
Taste	1	-	-
Oxidisability	1	-	-
Total Dissolved Solids	2	1500	1000
Chloride	2	250	250
Ammonia	2	1.5	1.5
Nitrat	2	10	10
Ferum/Iron	2	1.0	0.3
Fluoride	2	1.5	0.4 - 0.6
Hardness	2	500	500
Aluminium	2	-	0.2
Manganese	2	0.2	0.1
Chemical Oxygen Demand	2	10	-

Anionic Detergent MBAS	2	1.0	1.0
Biological Oxygen Demand	2	6	-
Nitrite	2	-	-
Total organic carbon (TOC)	2	-	-
Mercury	3	0.001	0.001
Cadmium	3	0.003	0.003
Arsenic	3	0.01	0.01
Cyanide	3	0.07	0.07
Plumbum/Lead	3	0.05	0.01
Chromium	3	0.05	0.05
Cuprum/Copper	3	1.0	1.0
Zinc	3	3	3
Natrium/Sodium	3	200	200
Sulphate	3	250	250
Selenium	3	0.01	0.01
Argentum	3	0.05	0.05
Magnesium	3	150	150
Mineral Oil	3	0.3	0.3
Chloroform	3	-	0.2
Bromoform	3	-	0.1
Dibromoklorometana	3	-	0.1
Bromodiklorometana	3	-	0.06
Fenol/Phenol	3	0.002	0.002
Antimony	3	-	0.005
Nickel	3	-	0.02
Dibromoacetonitrile	3	-	0.1
Dichloroacetic acid	3	-	0.05
Dichloroacetonitrile	3	-	0.09
Trichloroacetic acid	3	-	0.1
Trichloroacetonitrile	3	-	0.001
Trihalomethanes - Total	3	-	1.00
Aldrin / Dealdrin	4	0.00003	0.00003
DDT	4	0.002	0.002
Heptachlor & Heptachlor Epoxide	4	0.00003	0.00003
Methoxychlor	4	0.02	0.02
Lindane	4	0.002	0.002
Chlordane	4	0.0002	0.0002
Endosulfan	4	0.03	0.03

Hexachlorobenzena	4	0.001	0.001
1,2-dichloroethane	4	-	0.03
2,4,5-T	4	-	0.009
2,4,6-trichlorophenol	4	-	0.2
2,4-D	4	0.03	0.03
2,4-DB	4	-	0.09
2,4-dichlorophenol	4	-	0.09
Acrylamide	4	-	0.0005
Alachlor	4	-	0.02
Aldicarb	4	-	0.01
Benzene	4	-	0.01
Carbofuran	4	-	0.007
MCPA	4	-	0.002
Pendimethalin	4	-	0.02
Pentachlorophenol	4	-	0.009
Permethrin	4	-	0.02
Pesticides	4	-	-
Pesticides - Total	4	-	-
Polycyclic aromatic hydrocarbons	4	-	-
Propanil	4	-	0.02
Tetrachloroethene and Trichloroethene	4	-	-
Vinyl chloride	4	-	0.005
Gross alpha (α)	5	0.1Bq/l	0.1Bq/l
Gross beta (β)	5	1.0 Bq/l	1.0 Bq/l
Tritium	5	-	-
Total indicative dose	5	-	-

Appendix K: Water flow Pattern and Velocity profile for Model C

Step	Flow Pattern	Velocity
1		
2		
3		
4		

Appendix L: Characteristic Data

CHARACTERISTIC	1	2	3	4	5	6	7	8	9	10	11	12
Temperature, ^{O}C	30.8	30.51	30.9	30.8	30.8	30.7	30.95	30.91	30.9	30.81	30.9	31.2
pH	6.12	6.13	6.15	6.14	6.14	6.15	6.14	6.13	6.14	6.14	6.12	6.14
COLOUR PtCo (465nm)	-1	-1	-2	-2	-3	-1	-2	-2	-2	-1	-3	-4
TURBIDITY Ntu	10.54	6.13	10.12	12.1	14.76	7.51	11.6	13.1	6.12	10.53	11.23	11.53
CHEMICAL OXYGEN DEMAND [COD] mg/L	8	5.5	5.5	5.1	5.6	6.2	6.6	6.2	5.1	4.4	4	4.3
BIOCHEMICAL OXYGEN DEMAND [BOD] mg/L	2.1	2.3	1.7	1.7	1.8	2.4	1.9	2.3	2.4	1.62	1.7	2.2
TOTAL DISSOLVED SOLIDS [TDS] mg/L	132.21	140.1	141.2	131.41	144.62	130	136.55	142.33	143.11	147.55	144.21	132.1
SALINITY (ppt)	0.1	0.11	0.09	0.89	0.11	0.1	0.009	0.1	0.11	0.11	0.1	0.1
DISSOLVED OXYGEN ON SITE mg/L	1.5	1.5	1.55	1.56	1.52	1.6	1.51	1.54	1.56	1.55	1.57	1.55
CONDUCTIVITY uS/cm	233.56	235.16	222.91	222.8	244.12	234.15	246.5	245.91	234.45	239.1	245.3	234.66
Ferum (Fe) mg/L	11.1	11.4	11	11.2	11.1	11.1	11.3	11.5	11.3	11.2	11.1	11.4
Manganese (Mn) mg/L	0.55	0.52	0.5	0.53	0.51	0.53	0.54	0.53	0.56	0.52	0.53	0.54

CHAPTER ONE

INTRODUCTION

1.1 General

Groundwater is the water found between the fractures and space in soils, sand, and rocks. It is stored and moves slowly through the geological formation of soils, sand and rocks called aquifers. Groundwater is one of the sources of water, besides surface water, that can be used for daily needs, such as bathing, cooking, and washing, where it represents about 97% of the freshwater resources on the Earth that are available for human use (Lopez-Gunn and Jarvis, 2009). In Malaysia, however, the use of groundwater as a water source is minimal because most of its water sources are from surface waters such as rivers, lakes, and dams. In actual fact, many years ago, before Malaysia had systematic water distribution, people had used groundwater as a water source for their daily activities.

However, the increasing life expectancy, and technological capabilities reduce the use of groundwater as a water source as there are various technologies that allow the purification of existing water. Now, nevertheless, the use of groundwater is again being considered due to the contamination of surface water, but groundwater needs to be treated before its usage is expanded as one of the main water sources for humans. The main problem of using groundwater as a water source is the presence of minerals such as iron and manganese.

Iron and manganese are naturally-occurring minerals in the Earth's crust, where iron is the most widely discovered metal, which usually coexists with manganese. In drinking water, however, the World Health Organisation (WHO)

proposes that iron and manganese concentrations should be less than 0.3 mg/l and 0.1 mg/l respectively. Although the existing iron and minerals in drinking water are not health-threatening, it becomes a problem when they have contact with the bacteria found in soil, aquifers, and some surface waters (Rathinakumar *et al.*, 2014). The bacteria feed on the iron (Fe^{2+}) and manganese (Mn^{2+}) in water, consequently forming red-brown compounds for iron, or black-brown for manganese. This reaction is often detected in toilet tanks, pipe systems, and clogged water systems. The presence of iron and manganese in domestic drinking water delivery systems has become a serious problem because it changes the taste, colour, and odour of the water. According to Munter *et al.* (2005), iron behaviour depends on organic types and concentration, while organic substances (or silica) in water may interfere with the iron removal process by forming stable complexes with iron, Fe^{2+} and Fe^{3+}, with Fe^{3+} complexes being stronger and more stable compared to Fe^{2+}. In well water, the concentration of iron (Fe^{2+}) and manganese (Mn^{2+}) is seasonal, and varies with the depth and location of the well, and the geology of the area, where iron (Fe^{2+}) and manganese (Mn^{2+}) naturally occur in groundwater that has little or no oxygen (Kumar *et al.*, 2013).

The suitability of the method used to treat groundwater by removing the iron and manganese depends on the study area, soil type, and water characteristics as well as the operational costs, and amount of surface water. The study area plays an important role in determining the type of system to be used to remove iron and manganese as it requires a system that is economical, technically secure, and beneficial for the community (Wüthrich and Chanson, 2014).

To solve the issue of heavy metals in groundwater, a better technology has to be identified. The technologies used should be suitable with the raw water source, and the social and economic conditions of the surrounding community for the water treatment

to be truly effective. These technologies typically focus on the removal of iron and manganese from groundwater, which is accomplished by oxidation, precipitation, and sand filtration for the separation of the oxidation metals (Ellis *et al.*, 2000).

Conventionally, iron is removed from groundwater by the processes of aeration, and rapid filtration. Different mechanisms may contribute to iron removal in filters: flock filtration (Teunissen *et al.*, 2008), adsorptive iron removal (Teunissen *et al.*, 2008; Vries *et al.*, 2017), and biological iron removal (Juanjuan *et al.*, 2009; Yulan *et al.*, 2010). Water containing iron can be divided into two main groups: waters where iron is separated after aeration, and waters where iron remains in the solution after aeration for a long period of time (Munter *et al.*, 2005). Usually, roughing filters are primarily used to separate these fine solid particles of iron from the water that are only partly, or not at all retained by stilling basins or sedimentation tanks after the aeration process is completed. The large filter surface area available for sedimentation, and relatively small filtration rates also support absorption besides chemical and biological processes (Nkwonta and Ochieng, 2009).

In the design of a cascade aerator, in order to observe actual molecular reactions, software use is indispensable. However, there is no proper method of predicting the removal of iron and manganese in qualitative and empirical terms; the only way to do this is by studying expensive hydraulic models. Due to the high costs involved in the design and construction of a small-scale physical laboratory model, there is a need for further research in numerical simulation. The main disadvantage of the physical hydraulic model is the relatively long period of time required for building the model, data acquisition, and analysis. The numeric code introduced in the present study has no weakness. In this study, the use of the comprehensive LBM model is used with several cascade aerator designs, and comparisons with experimental development models will be discussed in detail. In addition, CFD and Avogadro is used in testing

the particles found in groundwater to see the reaction between the particles in this study.

1.2 Problem Statement

The main problem of this study area is that there are high concentrations of iron and manganese pollutants due to the periodically changing concentrations of water. Therefore, the raw water needed to be extracted directly from the existing tubes to flow into the cascade aerator to be treated. In order to make the cascade aerator more effective and improve the aerated groundwater quality, the oxygen transfer mass should be increased, which can be done by changing the cascade dimensions in the existing model, such as the height and angle of the cascade aerator. This is supported by Oh *et al.* (2015), where according to his study on mine water, the height of the mine drainage drop is a dominant factor in the efficiency of the cascade aerator, where the Fe^{2+} removal rate can be approximated by the prediction model with initial water quality summarised by the aerator drop height. This method increases the dissolved oxygen (DO), and decreases the concentration of iron and manganese without using any chemical products, while also ensuring that the water filter requires less maintenance. Therefore, the idea of increasing dissolved oxygen in the cascade aerator design is considered. However, the solubility of iron compounds increases at lower pH values. Usually, there is a difference between water-soluble Fe^{2+} and water-insoluble Fe^{3+} compounds.

Other aeration systems are less efficient as the cost for these systems is higher than the cost of the energy required to remove the heavy metals. In contrast, the proposed cascade aerator in this research will ensure maximum efficiency in removing the iron and manganese from the water source in all aspects, such as cost, air, and air

space. In addition, the high operation and maintenance costs of other types of aeration systems, which require high expertise for the maintenance of every part of the aerator operation, resulted in this method being used. Although the cascade aerator operating method is low cost, the aeration equipment works efficiently—without any solid waste floating on the water trapped in the cascade tank. This situation would affect the transfer of oxygen into the mass of water, as happens with other aerator systems.

Many techniques have been studied in water treatment structures. According to Bogue (2010), and Kumar *et al.* (2013), a cascade aerator is a tool that has a low construction cost, and is one of the most effective ways to treat groundwater. The Lattice Boltzmann Method (LBM) is used in this research because it deals with macro-scale problems related to fluid flow. In this research, LBM and experimental works are used together to investigate the aeration process in the newly designed cascade aerators. In designing new cascade aerators, different dimensions are used to determine the best design that can effectively decrease the concentration of iron and manganese. The LBM was chosen because this software is different from other Computational Fluid Dynamics (CFD) tools, such as Flown 3D, Mock Flow, and others, that are used in the School of Civil Engineering, USM. This software uses a code to design the model, and is very useful for modelling multiphase interfaces, and complicated boundary conditions.

Finally, the Dispersed Phase Method (DPM) and Avogadro Software are used to investigate the particles in the groundwater that affect the aeration process. The two software are used to investigate the interaction between the water particles, iron, and manganese. This interaction is important for this study because it provides information on the number of particles involved, how much the particles react, and how quickly the reaction can be stimulated. On the other hand, the reaction between the three particles (iron, manganese, and water) with oxygen may also be stimulated to prove

that the calculated equations can be demonstrated. The DPM considers particle detection in a simulation of various particle problems. The physical properties of discrete particles, velocity, and phase sizes are defined using constant phase conditions when the particles move through the flow. Both methods can help researchers in implementing the removal process more easily.

1.3 Research Objectives

This study embarks on the following objectives:

1. To determine the groundwater quality at Rumah Anak Yatim Nur Kasih, Taiping.
2. To investigate the performance of the two proposed laboratory-scale cascade aerator systems for water quality treatment.
3. To validate the performance of the cascade aerator systems using the Lattice Boltzmann Method (LBM).
4. To measure the removal of iron (Fe^{2+}) and manganese (Mn^{2+}) using Inductively Coupled Plasma (ICP), and validate them using particle-based Dispersed Phase Method (DPM) and Avogadro Software.

1.4 Scope of Work

The scope of study focuses on determining groundwater quality using standard laboratory tests and apparatus. The groundwater sample from the tube well at Rumah Anak Yatim Nur Kasih is analysed for pH, chemical oxygen demand, biological chemical demand, turbidity, colour, and heavy metal concentrations. There are two laboratory-scale models, namely Model A and Model B, with dimensions of 1.53 m (L) x 0.865 m (H) x 0.3 m (W), and 1.53 m (L) x 0.727 m (H) x 0.3 m (W). Their

parameters and operation performance are measured using the Inductively Coupled Plasma (ICP), portable spectrometer, submersible pump, flow sensor, turbidity meter, Arduino, and ICP standard solution.

Simulation of the performance of the cascade aerator system is carried out based on the Lattice Boltzmann Method. The results are viewed using the Paraview Software. Validation between the experiment and computer simulations is done using particle image velocimetry. Further verification of iron and manganese in groundwater is through the Discrete Phase Model. The interaction between the water molecules, iron, and manganese is visualised using Avogadro Software. Determination of optimum pH for oxidation is conducted with the Geochemist's Workbench. All the software used in the study are the Lattice Boltzmann, PALABOS, ANSYS-DPM, Geochemist's Workbench, and Avogadro. Some of the challenges in carrying out this study are the large models which need to be transferred to the field, and limited storage space at the orphanage.

1.5 Thesis Organisation

This section briefly outlines the content of each of the five chapters in this thesis. Chapter 1 consists of an introduction to the research, problem statement, the objectives and the scope of work for this research. Chapter 2 presents the review of previous research related or associated with the present research. The review includes some recent works on iron and manganese, cascade aerators, aeration, software as well as experiments. Chapter 3 presents the methodology that provides information on the flow of research, and briefly introduces the site of study for this research. Chapter 4 presents the numerical method, the procedure used in simulating the cascade aerator model, and the software used throughout the research. It also presents the data collected on site, and the simulation data. Subsequently, all the data are analysed and

discussed in this chapter. Finally, Chapter 5 concludes this thesis in parallel to the research objectives stated in Chapter 1, closing with several recommendations for future studies.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter discusses previous research, and fundamentals regarding the cascade aerator, aeration, simulations, experiments, and validation methods. This chapter also discusses the geometry and dimensions of the cascade aerator that should be modelled based on previous studies. This work is different from others because the cascade aerator is used to treat groundwater, and not surface water or wastewater. The study of the literature is done to find the ideal dimensions to redesign the cascade aerator to effectively remove iron and manganese.

Groundwater plays an important role in the development of water resource management. Therefore, there is a growing demand for hydrological information on groundwater, and the hydraulic movement of water in aquifers. The main purpose is to ensure that the use of groundwater has its advantages, and can be treated to become a well-preserved water source. One of the ways to treat groundwater is through the aeration process, in which oxygen is very important. Hence, the amount of oxygen dissolved in the groundwater should be calculated to make sure that the contaminants in the groundwater are removed before the water is supplied to the consumer. The method selected to improve the oxygen content in the aeration process to eliminate groundwater pollution is by using the cascade aerator.

There are many factors why the cascade aerator was chosen for this study, which will be explained in detail. Additionally, the use of numerical studies in the implementation of this study is helpful in getting the best results. There are many studies on the use of Computational Fluid Dynamics (CFD) Software, and a few

studies that use the Lattice Boltzmann Method (LBM) in modelling and simulating the cascade aerator. On the other hand, many methods have been developed to investigate the aeration process, such as Volume of Fluid Method, Flown 3D model, laboratory experiments, and others.

Therefore, the literature review here focuses on the fundamentals of the aeration process, and the LBM applied by the researcher in designing the cascade aerator.

2.2 Groundwater

Groundwater is water that flows or collects beneath the Earth's surface, and originates from rain, and melting snow and ice. It sinks into the ground, filling the small empty spaces in soil, sediments, and porous rocks. Aquifers, springs, and wells are supplied by the flow of groundwater (Bouchard *et al.*, 2011; Katsoyiannis and Zouboulis, 2004; Yulan *et al.*, 2010). Appelo *et al.* (1999) conducted a study on the removal of iron and manganese from groundwater by using in situ modelling, where the modelling used a volume of oxygenated water, and a large volume of groundwater. The result of the experiment showed that the concentration of oxidants in injected water has an insignificant effect due to low changes in the ferrous iron.

The problem for researchers is on how to inject maximum Dissolved Oxygen (DO) in the water, and achieve safe drinking water standards for groundwater. SPSS analysis software has been used to determine the concentration of dissolved oxygen in the removal of iron and manganese (Juanjuan *et al.*, 2009). Ellis *et al.* (2000) focused on the microfiltration (MF) of iron and manganese with variables such as tangential flow rate, pressure, and metal feed concentrations, where the artificial and natural groundwater showed similar behaviours. In Morocco, ferrous iron in groundwater was

studied by Azher *et al.* (2008), where they found that the mixing reactor was justified, and the iron was oxidised from the aeration process in the 63 L split-rectangular airlift reactor.

In 1998, in order to overcome the dry season and expand urban water supply, the Malaysian federal government announced in the Ninth Malaysia Plan (2006-2010), "...groundwater development will be promoted as an interim measure to address water shortages in Selangor, Kuala Lumpur and Putrajaya" (EPU, 2005).

Groundwater is often seen as a reliable source of clean water, making it an ideal source for the water demand in urban areas. But in urban areas, in particular, aquifers are often threatened by pollution and over extraction that can destroy these groundwater sources. To protect groundwater resources, Health and Safety Regulations were implemented in the United States in the 1920s. In Kuala Lumpur, a lack of knowledge on groundwater has been highlighted in some water resource studies, but there is no underground monitoring and business modelling. However, there is an increase in incentives to make groundwater a potential drinking water source for Kuala Lumpur including the increase in water demand caused by population growth, economic threats, and pollution to lakes and rivers. Therefore, it is timely to re-evaluate the potential of potable groundwater as a drinking water source for Kuala Lumpur. In fact, groundwater is used in Kedah and Kelantan as an alternative to tap water in low water pressure areas, and in rural areas. However, the use of groundwater is not as extensive as the use of surface water.

A pilot plant for groundwater in northern Croatia was studied by Štembal *et al.* (2005). The study focused on removing ammonia, iron, and manganese from the groundwater, where nitrification was only detected in the middle part of the biofilter; however, the iron, ammonia, and manganese had disappeared completely. Besides

that, a comparison of full-scale trickling filters used in the Oasen water treatment plant, Lekkerkerk, showed some differences in the removal of manganese. There were some problems with the combination of nitrification, where the nitrification encouraged competition between phosphate and essential trace substrates in biological processes due to the incomplete removal of manganese (de Vet *et al.*, 2010). Meanwhile, Berbenni *et al.* (2000) had conducted a laboratory test to prove that biological processes and autocatalysis play a role in the elimination of manganese in the liquid phase, and the whole process depended on parameters such as redox potential, temperature, and sludge age.

Figure 2.1 shows the major physicochemical changes and redox reactions occurring along the groundwater flow paths when a confined groundwater system is entered by groundwater. According to widely used models for closed oxidation systems, spatial distribution of the oxidised species in closed aquifers containing excess dissolved oxygen concentrations (DOCs) should follow predictable patterns. The input of the further oxidised species will be closed, and the expected reductions will occur along the path (Malard and Hervant, 1999). The first reaction, due to the high free energy change (± 120 kcal), is aerobic breathing, resulting in the loss of DO in groundwater. This model is used to answer two questions: the extent to which the distance from the oxygen reaction area is lost, and how long the oxygen has been in the groundwater. Both of these questions need to be answered to produce effective models for closed oxidation systems. Based on studies conducted in Malaysia and abroad, groundwater is used to replace contaminated water resources or surface water. In any case, the use of groundwater should be studied in terms of its properties and contents that allow water treatment to be carried out before being channelled to consumers.

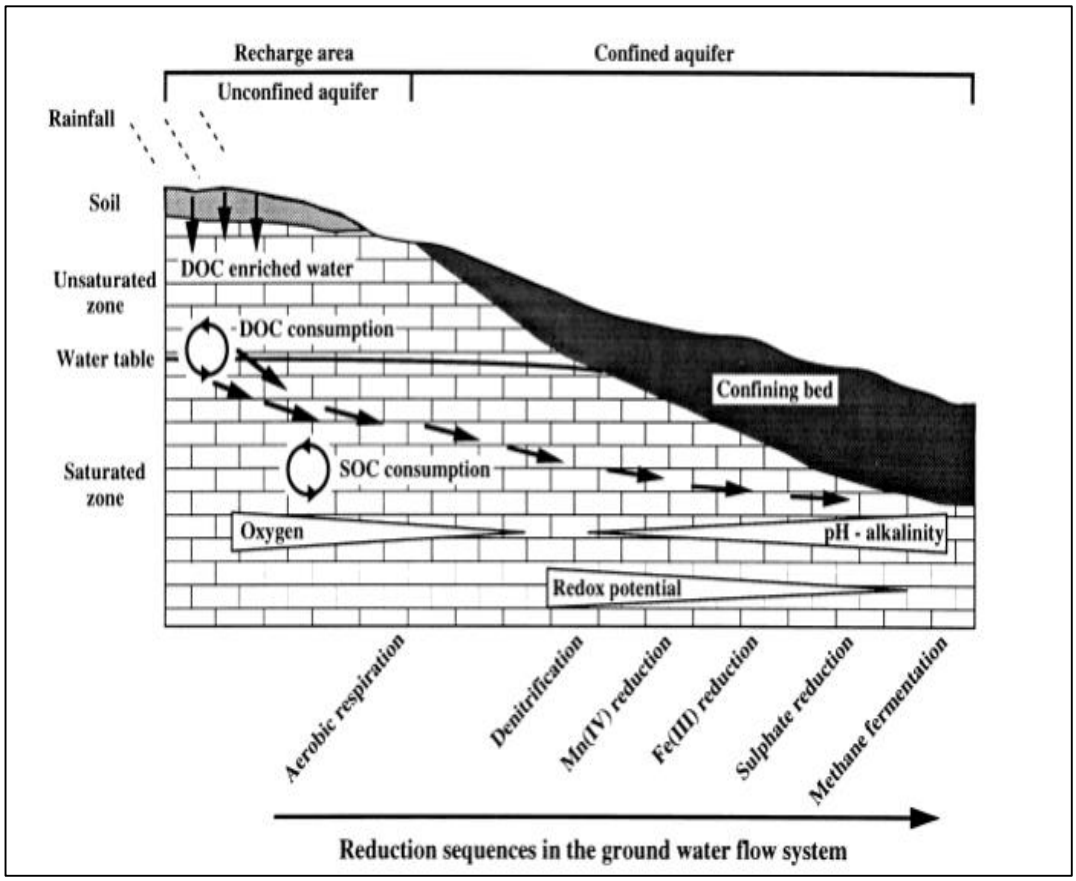


Figure 2.1: Major physicochemical changes and redox reactions occurring along groundwater flow paths in a confined aquifer system (Malard and Hervant, 1999)

2.3 Heavy Metals

Heavy metals are defined as metals with high densities, atomic numbers, or atomic weights, according to the explanation by Hawkes (1997), which are metals of high specific gravity, especially those having the specific gravity of 5.0, or densities above 5 g/cm^3. From a chemistry definition, there are no metals with densities less than 5 g/cm^3, but this parameter is of little concern to chemists compared to the metals' chemical properties or behaviours. Common heavy metals such as copper, iron, silver, gold, and many more can be discovered in the ground. However, heavy metals found in the soil will contaminate the groundwater. Contamination occurs when solid waste from industrial units, which is disposed of near the factories, react with percolating rain water, and reach groundwater. Water absorption into the soil will collect a large

amount of heavy metals that reach the aquifer system, and pollute the groundwater. The use of water contaminated from mercury, arsenic, and cadmium, which are used or produced by many industries in the mainland, causes illnesses. Soil may be contaminated by the accumulation of heavy metals and metalloids through emissions from a rapidly growing industrial area (Wuana and Okieimen, 2011). The heavy metal pollution of the land can pose risks and dangers to humans and ecosystems through: direct dialling or contact with contaminated soil, the food chain (human-grown plants → humans and animals), and contaminated groundwater. However, there are also commonly used procedures for the removal of metal ions from aqueous liquids including chemical precipitation, inverse osmosis, and solvent extraction, but these methods have weaknesses such as incomplete metal removal, high reagent and energy requirements, toxic sludge generation, or other waste products that require disposal (Chandra Sekhar *et al.*, 2003).

Heavy metals can also be hazardous because of their higher tendency to increase concentration in biological organisms compared to the chemical concentration in the environment. This is because, in the environment, heavy metals are formed alone without active reaction. Therefore, Parameter Limits for Standard A and Standard B by the Department of Environment (DOE) Malaysia, shown in Table 2.1, are required to determine the standard to be followed. These standards should be used and followed by industries that produce heavy metals, where they should dispose of them responsibly instead of releasing the heavy metals into rivers or lakes. This is important in preventing water sources from being contaminated and causing various illnesses and side effects.

Table 2.1: Parameter limits for Standard A and Standard B (Source: Department of Environment (DOE))

Parameter	Unit	Standard A	Standard B
Lead (II)	mg/l	0.10	0.50
Cadmium (II)	mg/l	0.01	0.02
Manganese (II)	mg/l	0.20	1.00
Nickel (II)	mg/l	0.20	1.00
Zinc (II)	mg/l	2.00	2.00
Ferum (II)	mg/l	1.00	5.00

According to Rosman (2010), heavy metals are chemical elements that are five times the specific gravity of water. The specific gravity of heavy metals is measured using the density of a given amount of a solid substance compared to an equal amount of water. Some examples of heavy metals that have a higher specific gravity than water are iron (7.9), manganese (7.42), nickel (8.9), mercury (5.7), and lead (11.34).

2.3.1 Toxicity of Heavy Metals

There are more than 15 heavy metals recorded, but only four are detrimental to human health: mercury (Hg), cadmium (Cd), lead (Pb), and inorganic arsenic (As). The presence of these heavy metals found in nature is toxic to humans, proven by the health problems arising from exposure to heavy metals, hence making them a major threat to human health (Jaishankar *et al.*, 2014). According to Wynne (2010), these four heavy metals are always present in toxic waste sites. The high toxicity of heavy metals can be damaging even in very low concentrations as they are stored in the kidneys, and hard tissues such as bone (Rosman, 2010).

2.3.2 Iron and Manganese

Iron and manganese are two of the heavy metal elements that can be found in groundwater. They can be detected through two ways: observation (when iron and manganese are exposed to oxygen, colour changes occur, where iron is oxidised into a red-brown compound while manganese turns black-brown), and tests performed at the laboratory that determine the concentration of the metals. They are mostly present in the soluble form of divalent iron (Fe^{2+}) and manganese (Mn^{2+}) ions (Khadse *et al.*, 2015). Additionally, according to Khadse *et al*. (2015), waters containing iron and manganese exposed to air or oxygen will become cloudy and turbid due to the oxidation of iron and manganese to Fe^{3+} and Mn^{4+}, which form colloidal precipitates. This is due to the rapid reaction between the heavy metals and air when heavy metals are exposed to air. Therefore, it only takes a short amount of time for the form of the heavy metals to change to Fe^{3+} or Mn^{4+} after being exposed to air (Sawyer, 1959).

In other countries, there are two tests conducted by researchers to test public water supplies, and private water supplies (Bruce and Sharon, 2014) according to the U.S. Environment Protection Agency (EPA), which fall into either of two categories: Secondary Standards, or Primary Standards. Secondary Standards are based on aesthetic factors such as taste, colour, appearance, staining, and others. On the other hand, Primary Standards are based on health considerations, and are designed to protect human health. Similarly, testing of private water supplies needs to be done in laboratories, and in accordance with EPA methods.

Testing the water treatment is a method to clean public, and private water supplies. Sometimes, the iron in pipes could cause the appearance of iron in the water supply, according to Bruce and Sharon (2014). In deep wells where oxygen content is low, the iron/manganese-bearing water is clean and colourless. This means that if the

iron and manganese are not exposed to air, then changes in taste and appearance, and staining would not occur.

2.4 Techniques Used to Remove Iron and Manganese

There are many types of techniques in removing iron and manganese from the domestic water system. In aeration systems, there are several ways that are used to remove iron and manganese, such as the biological aerated filter (Yulan *et al.*, 2010), oxidation (Munter *et al*., 2005), and cascade aeration (Kokila and Divya, 2015). These systems are usually used in actual industries in order to remove iron and manganese in surface water, groundwater, and wastewater treatment. Table 2.2 below shows the iron and manganese removal methods and description.

In the treatment of passive manganese, manganese can now be removed at the same time as iron, and manganese removal is efficient. This condition can be implemented with the catalytic action of this substrate coupled with aeration, providing the conditions necessary to overcome the kinetically slow manganese oxidation in the presence of dissolved iron (Johnson, 2003). Aeration interests have been demonstrated, especially when the system is subject to environmental stresses such as low (or non-existent) light, low temperatures (down to 4°C), and the presence of dissolved iron in influential water. If iron concentrations are high, both manganese and iron can be eliminated.

The use of biofiltration (as in biological aerated filters) has been studied by Burger *et al.* (2008), where their study with small columns that have been injected with indigenous biofilms from manganese (Mn) filtration plants, and filtration columns that have been injected with a liquid suspension of Leptothrix discophora SP-6 showed that the removal of manganese can be done with a larger pH range than what they had

previously investigated. But the ability of this treatment technology to work extensively in influential conditions allows more people to consider biological treatments as an option to eliminate manganese from their drinking water (Burger *et al*., 2008).

In terms of chemical reactions, removal of iron and manganese from lakes using chlorine dosage has been investigated using ultrafiltration (UF) systems. In this system, chlorine is added to remove iron and manganese from drinking water (Choo *et al.*, 2005). In Choo *et al.*'s (2005) study, the use of chlorine depended on the content of iron and manganese in the water source. The addition of chlorine into the water increased the efficiency of manganese removal, whereby the process occurred rapidly until it reached 80% removal using a dose of chlorine of 3 mg/l, known as Cl_2.

Table 2.2: Iron and manganese removal methods

Method	**Description**
Biological Aerated Filter	Biological aerated filters use two layers of ceramsite as support materials in the lab. The system is simultaneously injected with iron bacteria and manganese oxide, and nitrifying bacteria, and a series of experiments is carried out to investigate the removal of simultaneous filters. The effect of variation in the rate of simultaneous removal of nitrogen, manganese, iron, and ammonia is studied. At the same time, the effluent concentration and removal of ammonium, manganese, iron, and nitrogen along the depth of the filter is analysed.
Oxidation	Oxidation is the loss of electrons during reaction by molecules, atoms, or ions. Oxidation occurs when the oxidation state of the molecule, atom, or ion increases. This happens in the case of iron and manganese where the ferrous system acts as a catalyst for the transfer of electrons to the oxidation of organic matter, depending on the Fe^{2+} oxidation and Fe^{3+} reduction rates compared to rate of total oxygen reduction of Fe^{2+} by organic matter. The higher the pH, the more the oxidation process of more Fe^{2+} organic complexes is retarded. For example, at pH 8, the reduction of the constant rate by a factor of 10 results in doubling the semi-life of complex Fe^{2+} with oxidation. Oxidation will affect the removal of substances such as ammonia, carbon dioxide chlorine, hydrogen sulphide, methane, iron, and manganese.
Cascade Aeration	Cascade aerators drain air into waterways to oxidise iron, and reduce dissolved gases. This solution carries water and air in close contact to remove dissolved gases (such as carbon dioxide). It oxidises dissolved metals such as iron, hydrogen sulphide, and volatile organic chemicals (VOCs).

In addition, according to Lin *et al.* (2011)**,** Palygorskite can also be used in the removal of iron and manganese in the form of aqueous solution. Chemicals and reagents were used in conducting batch studies where the combustion temperature, the effect of reaction time, and acid concentration were monitored. As a result, both the metal iron and manganese had been exposed well, and achieved equilibrium absorption in a short period of time (Lin *et al.*, 2011).

In wetland treatment systems, the removal of iron and manganese is reviewed through rates, processes, and implications for management, where the removal of iron is successfully done with the installation of a passive treatment system. However, if the concentration of iron is very low, the biotic removal process will become a priority (Batty *et al.*, 2008). According to Batty *et al.* (2008), manganese can be eliminated using the wetland, but the removal of iron must be done before the manganese removal takes place. This indicates that the process of removal of iron and manganese cannot be done simultaneously. The following subtopics describe four other methods of iron and manganese removal (namely, biological aerated filters, biosand filters, freeze/thaw cycles, and chemical reactions) aside from aeration, which is the focus of the current study.

2.4.1 Biological Aerated Filter

A biological aerated filter (BAF) is a filter that has 2 to 3 metres of small-sized filter media in order to provide a high surface area to grow biomass. BAF is used to treat groundwater containing high concentrations of iron, manganese, and ammonia nitrogen at low temperatures. A study was conducted by Ma *et al.* (2010) on the high concentrations of heavy metals and the effects of contaminants in raw water, where the readings were 2.52 to 4.22 mg/l, 1.22 to 3.97 mg/l, and 1.24 to 3.92 mg/l. The

research results showed that the biological oxidation of iron and manganese had achieved a steady state, while the nitrification process was affected by hydraulic loading. In addition, iron, manganese, and ammonia nitrogen could be removed simultaneously under hydraulic loading when the iron inlet concentration is low (Ma *et al.*, 2010).

Moreover, in Piao *et al.* (2011)'s study, it was found that for groundwater that was treated with aeration, and contact oxidation filtration for the removal of iron and manganese pollutants, the removal rate of organic material and ammonia nitrogen from the polluted water was low. Concurrent biology was also practised to remove iron, manganese, and ammonia from contaminated groundwater. Based on previous studies, the use of biological aerated filters can influence the rate of simultaneous removal of manganese, iron, and ammonia nitrogen, in which the filter comprises two layers as reinforcement material in the laboratory (Tekerlekopoulou and Vayenas, 2007; Yulan *et al.*, 2010).

Nowadays, the biological treatment process is more commonly used than the chemical treatment process because it has no need for addition of extra chemicals, requires low-cost maintenance, and has a high filtration rate. Pilot-scale BAF consists of Plexiglas, with a 3-metre height and 185 cm external diameter, where the operations of the BAF column will run for several months. When the material turns black, this means that manganese oxidation has occurred. Raw water in the head tank flows across a flowmeter, and drops onto the biofilter layer through the perforated pipe at the top of the filter, and then flows directly to the bottom of the filter that is perforated by a pump. According to Ma *et al.* (2010), their study provided a 0.8 metre-thick layer of filter coating with a mixture, with a 3.2 to 5.0 mm diameter at the top, and a 1.6 to 3.2 mm diameter at the bottom. This material combination provides advantages to effectively produce more dissolved oxygen at the top of the water. Figure 2.2 shows

the schematic drawing of the experiment setup of the BAF where: 1. Head tank 2. Water balance tank 3. Liquid flowmeter 4. Perforated water pipe 5. Filter column 6. Filter media of ceramic beads 7. Sampling point 8. Supporting layer 9. Effluent pipe 10. Air pump 11. Air flowmeter 12. Perforated air pipe 13. Backwashing water pipe 14. Valves 15. Overflow pipe.

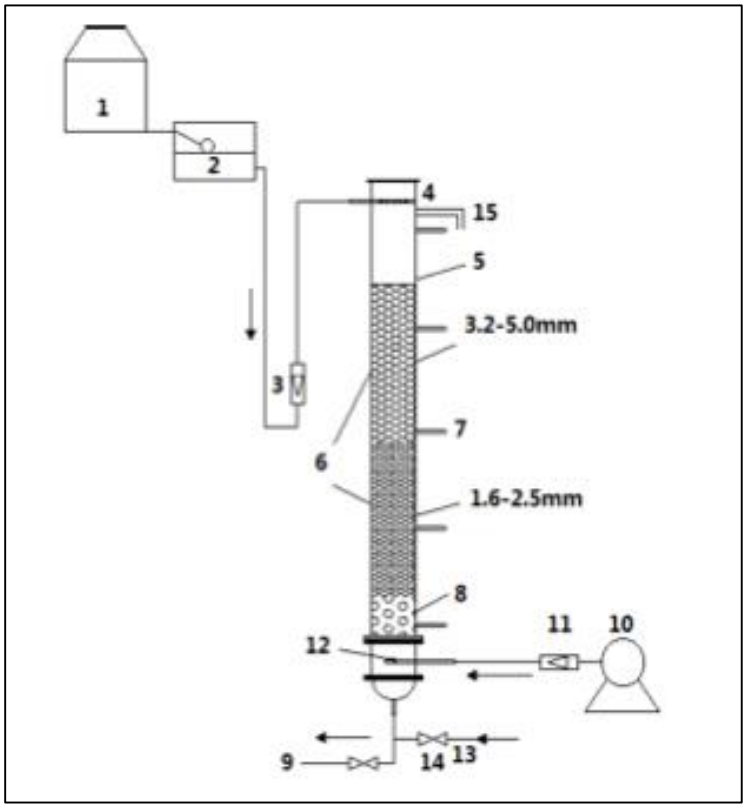


Figure 2.2: Schematic drawing of the experiment setup of BAF (Ma *et al.*, 2010)

2.4.2 Chemical Reactions

In aerated water, the potential of redox water allows the oxidation of ferrous ferrite into ferrous iron. This condition results in precipitate, which then accumulates into hydroxide iron, $Fe(OH)_3$. The conversion causes the removal of dissolved iron to be done naturally. This can be seen in the reaction equations below:

$$4\ Fe_2 + 3\ O_2 \dashrightarrow 2\ Fe_2O_3 \quad (2.1)$$

$$Fe_2O_3 + 3\ H_2O \dashrightarrow 2Fe\ (OH)_3 \quad (2.2)$$

Iron reactions in water depend on the potential of pH and redox. Usually, groundwater has a low pH, between pH 5.5 to 6.5, and has a low oxygen content.

The same goes for manganese in water, where its presence presents danger neither to human health nor the environment, but is unpleasant. Manganese will undergo oxidation, similar to iron, from Mn^{2+} to Mn^{4+}, which precipitates to form manganese dioxide (MnO_2).

$$Mn^{2+} \xrightarrow[\text{oxidation}]{} Mn^{4+} \xrightarrow[\text{precipitation}]{} MnO_2 \qquad (2.3)$$

2.5 Aeration

Aeration is the process of increasing or maintaining the oxygen saturation of water in both natural and artificial environments. Aeration is important to provide enough dissolved oxygen (DO) for the aerobics of organisms in biochemical oxygen demand (BOD) removal and nitrification in activated sludge plants, and to retain contaminants in suspension biomass (Amand *et al.*, 2013). To treat groundwater, aeration methods can also be used. It was found, through a study conducted in West Netherland, that the nutrient-rich groundwater can impose heavy phosphate loads on surface water systems. Iron oxidation will oxidise phosphate (PO_4) quickly at neutral pH, and PO_4 may also be bonded to calcium (Ca) precipitates at increased pH, so the estimated load based on conservative behaviour during exfiltration will be too large (Griffioen, 2006). According to Griffioen (2006), there are three models to describe the rapid binding of PO_4 by iron oxide phase formed with oxygenation by diffusing Fe^{2+}, each based on different concepts. These concepts are surface complexes, solid solutions, and two-mineral precipitation. This model is useful for calculating PO_4 surface water loads from underground exfiltration, taking into account the rapid flow

(<1 day) during exfiltration. In addition, experiments were performed by Daud *et al*. (2013) to eliminate the concentrations of iron and manganese in groundwater through two-way aeration methods (water to air, and air into water), followed by process filtering using manganese greensand. The characteristics of groundwater, such as pH, dissolved oxygen, turbidity, and concentration of heavy metals (iron and manganese) was evaluated when dissolved oxygen content showed an increase, and all iron and manganese removal reached the standard 0.3 mg/l and 0.1 mg/l. From the study, it could be concluded that the method of aeration of air-into-water gives a higher percentage of iron and manganese removal compared to water-to-air.

According to Alp and Melching (2011), aeration is defined as using atmospheric air as an oxygen source to build oxygenation. This is an important process in water and wastewater treatment—transferring oxygen from the gas to liquid phase between the atmosphere and water (Moulick *et al.*, 2010; Baylar *et al.*, 2007a). Aeration brings water and air into close contact to remove dissolved gases (such as carbon dioxide), and oxidise dissolved metals such as iron, hydrogen sulphide, and volatile organic chemicals (VOCs). Therefore, supplemental aeration could be an effective approach to improve DO concentration (Baylar *et al.*, 2007a; Alp and Melching, 2011; Abu Hasan *et al.,* 2014). Aquatic life in rivers and streams need aeration for respiration and life continuity; hence, the survival of the aquatic life is dependent on the aeration system. According to Baylar *et al*. (2009), there are several hydraulic structures used in the aeration process, such as water jet aeration with circular nozzles, water jet aeration with venturi nozzles, pipe aeration with venturi tubes, high-head conduit aeration, weir aeration, and free-surface conduit aeration.

Demars and Manson (2012) studied the accurate estimation of gas transfer in streams. This study referred to the Dobbins theory which theorises an interfacial film that is assumed to exist in the water. In this theory, these interfacial films are assumed

to exist in the composition of the liquid below the surface which will be replaced at random. Thus, according to the Dobbins theory, the gas velocity response to temperature variations, which affects the properties of water and molecular diffusion, is not important as the aggression rapidly renews the boundary layer of the focus on the air interface. However, other theoretical models differ widely in the response to temperature.

According to Moulick *et al.* (2010), aeration is a critical process in water and wastewater treatment as it transfers the oxygen from gaseous to liquid form. Three methods that are commonly used for aeration are: gravity aeration, mechanical aeration, and water-diffused aeration (Moulick *et al.*, 2010). A simple weir is utilised for gravity aeration, which involves an inclined corrugated sheet, or stepped cascade. Sniders and Laizans (2011) studied a wastewater aeration tank model with built-in equipment for uniform distribution of atmospheric air. The data was analysed using the Mathematical Modelling for wastewater aeration, and Simulink Modelling for oxygen concentration transient processes. Mathematical Modelling is analysed based on linear stationary models of wastewater aeration, and block diagram for comparative research of stationary and non-stationary models, while the oxygen concentration transient process consists of the development of adaptive self-turning virtual models (Sniders and Laizans, 2011). The block diagram showing the dissolved oxygen in stationary and non-stationary models is in Figure 2.3.