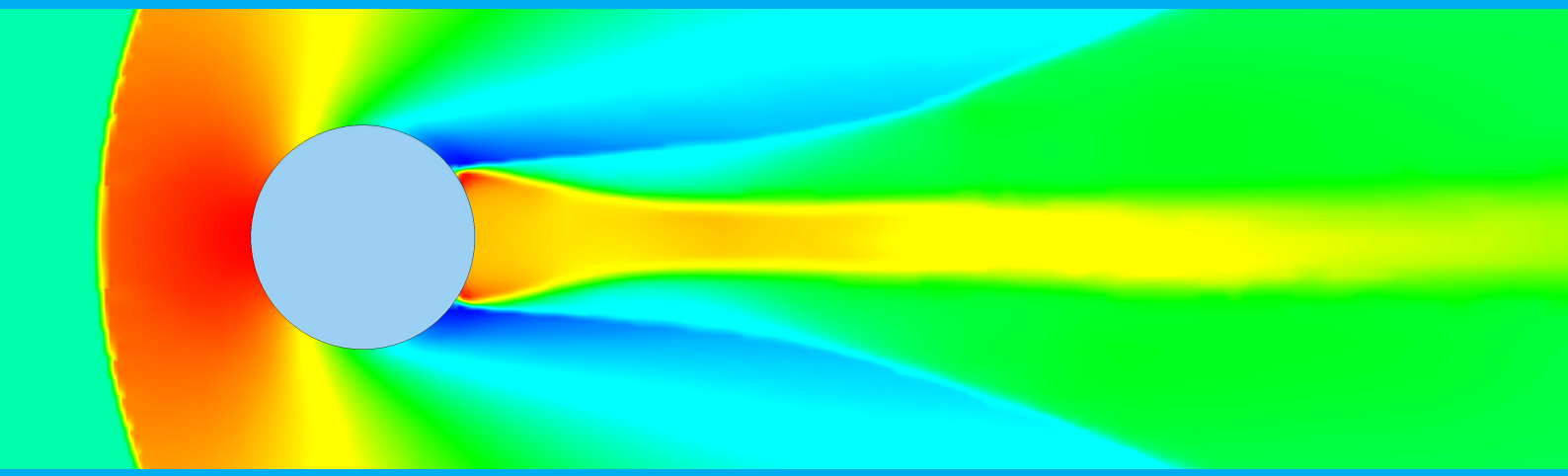
A C++11 Implementation of a Moving Wall in the Lattice Boltzmann Method

Computational Fluid Dynamics to determine the viscosity of Molten Thorium Fuel Salt in Molten Salt Reactors

J. Seelen

Thesis for the fulfilment of the Degree,
Master of Science in Sustainable Energy Technology,
in the Field of Nuclear Energy,
at Delft University of Technology



TU Delft

A C++11 Implementation of a Moving Wall in the Lattice Boltzmann Method

Computational Fluid Dynamics to determine the viscosity of Molten Thorium Fuel Salt in Molten Salt Reactors

by

J. Seelen

Thesis for the fulfilment of the Degree,
Master of Science in Sustainable Energy Technology,
in the Field of Nuclear Energy,
at Delft University of Technology
to be defended publicly on Wednesday December 21, 2016 at 10:30 AM.

Student number:	1317547
Project duration:	Februari 1, 2016 – December 21, 2016
Thesis committee:	
Prof. dr. ir. J. L. Kloosterman	Professor Department of Radiation Science & Technology head of section Nuclear Energy and Radiation Applications
Dr. M. Rohde	Assistant Professor Department of Radiation Science & Technology section Physics of Nuclear Reactors Thesis Supervisor
Dr. M. Möller	Assistant Professor Department of Applied Mathematics section Numerical Analysis

This thesis is confidential and cannot be made public until December 21, 2016.
An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

After my BSc in Aerospace Engineering I started my Master Sustainable Energy Technology to broaden my skill set. Aerospace Engineering gave me an interest in Computational Fluid Dynamics and programming has always been a hobby of mine. Hence this thesis topic interested me immediately.

The purpose of this thesis is to contribute to the study on Computational Fluid Dynamics with the use of the Lattice Boltzmann Method by analysing the best way to simulate a fuel salt with use of C++ and giving advise on how to go further.

I would like to thank Martin Rohde for his guidance and advice during this entire project. I would also like to thank Jan-Leen Kloosterman and Matthias Möller for being part of my committee and checking my work.

This thesis could not have been possible without the financial help from the “Stichting Fundatie van de Vrijvrouwe Renswoude te ’s-Gravenhage”, “Schuurman Schimmel van Outeren Stichting” and my two uncles Jan Vrakking and Cees Seelen.

I hope you enjoy reading this thesis and the knowledge gained from this thesis will benefit future research in fuel salt simulation.

J. Seelen
Delft, December 18, 2016

Abstract

This thesis is part of the research on the thermodynamic properties of thorium salt and aims to provide a C++11 code to simulate fuel salt flow at various temperatures around a settling sphere and inside a container. This thesis should form the basis of a C++11 implementation on fuel salt flow in the Molten Salt Fast Reactor using the Lattice Boltzmann Method. The research into the thermodynamic properties of the thorium fuel salt is necessary in the research and development of new Generation IV Molten Salt Fast Reactors. The simulation uses the Lattice Boltzmann Method in the C++11 code language and the Palabos open-source library which are introduced in chapter 1.

The governing equations in this simulation are the dynamics of the sphere, the Lattice Boltzmann Equation (LBE) and the boundary method equations as presented in chapter 2. The boundary condition chosen for this thesis is the Guo extrapolation method with the Inamuro iteration to determine the fluid force and torque on the sphere. These models were chosen since they existed in the Palabos library and were used in 3D moving wall examples. Since there was no documentation available from Palabos on the off-lattice 3D methods, building the model proved to be time consuming and was a trial and error process. The resulting code as shown by chapter 3 is still a work in progress. The challenge was to combine both a settling sphere or in other words a moving wall with fluid wall interactions, in 3D and inside an enclosed container (hence no infinite flow in any direction and a second boundary).

Due to the complex nature of the problem and the lack of documentation, time ran out before results could be obtained. If the model would have generated results these would have needed to be verified and validated as presented by chapter 4. The intended simulation presented in chapter 5 was a settling sphere inside a cylindrical container with simulations for varying temperatures and Reynolds numbers. As stated earlier the model did not obtain results but the project did as stated in chapter 6, namely a good understanding of modelling fuels salts with C++ and the Lattice Boltzmann Method (LBM) Palabos library is obtained. Therefore in chapter 7 it is recommended: to either update the palabos library and expand the documentation, resulting in a library owned and maintained by Delft University of Technology for Computational Fluid Dynamics (CFD) with the Lattice Boltzmann Method (LBM) in C++, or choose a different library entirely. Any such library should make the most of the advantage that the LBM has, namely that it is a very parallel method. Proper documentation and coding standards should be implemented keeping the library readable and understandable. Hence the conclusion in chapter 8 is that the Palabos library might prove a good solution in the future if modified and documented. More research on different boundary conditions and their performance is required. More research is also required on the performance of the Palabos library compared to other libraries. Finally a performance analysis for hardware and parallelisation methods would be required to make a decision on improvements in that department.

Definitions

Notation	Description	Page List
Lanthanides	any of the series of fifteen metallic elements from Lanthanum to lutetium in the periodic table. Atomic numbers 57 through 71	1, 4
Palabos	An open-source C++ library of Computational Fluid Dynamics in the Lattice Boltzmann Method www.palabos.org	1, 2, 23, 27, 28, 30, 47
Thorium	Is an element with symbol **Th** and atomic number 90	1, 2, 4

Acronyms

Notation	Description	Page List
BGK	Bhatnagar Gross Krook	1, 20, 21
CA	Cellular Automata	1, 12
CFD	Computational Fluid Dynamics	v, xiv, 1, 9–11, 14, 15
DEMO	Department of Electronic and Mechanical Development (Dienst Elektronische en Mechanische Ontwikkeling)	1, 9
e.g.	exempli gratia	1, 2, 44
EU	European Union	1, 2
FDM	Finite Difference Method	1, 11
FEM	Finite Element Method	1, 11
FHR	Fluoride salt cooled High-temperature Reactor	1, 2
FVM	Finite Volume Method	1, 11
GDB	GNU DeBugger	1, 37
GIF	Generation IV International Forum	1, 2, 8
GPGPU	General Purpose computing on Graphical Processing Units	1, 47
HPC	High Performance Computing	1, 29, 30
LBE	Lattice Boltzmann Equation	v, 1, 13, 19
LBGK	Lattice Boltzmann Bhatnagar Gross Krook Method	1, 21, 22
LBM	Lattice Boltzmann Method	v, 1, 2, 10–12, 14–17, 27, 33, 46, 47
LES	Large Eddy Simulation	1
LGA	Lattice Gas Automata	1, 12
MPI	Message Passing Interface	1, 28, 29, 31
MRT	Multiple Relaxation Time	1, 20
MSFR	Molten Salt Fast Reactor	xiv, 1–5, 7
MSR	Molten Salt Reactor	1, 2, 5
NERA	Nuclear Energy and Radiation Applications	1, 2, 6, 8
PID	Proces IDentification	1, 37

Notation	Description	Page List
RID	Reactor Institute Delft	1, 47
SAMOFAR	Safety Assessment of the Molten Salt Fast Reactor	1, 2, 8
SET	Sustainable Energy Technology	1
SRT	Single Relaxation Time	1, 20
STL	STereoLithographic	1, 31, 32, 43, 44
VTK	Visualisation ToolKit	1, 34

List of Symbols

Notation	Description	Symbol	Page List
CloKMag	A molten salt containing Kalium Chloride and Magnesium Chloride	$KCl\text{-}MgCl_2$	1, 5
FLiBe	A molten salt containing 2 Lithium Fluoride and Beryllium Fluoride	$2LiF\text{-}BeF_2$	1, 5
FLiNaBe	A molten salt containing Lithium Fluoride, Natrium Fluoride and Beryllium Fluoride	$LiF\text{-}NaF\text{-}BeF_2$	1, 5
FLiNaK	A molten salt containing Lithium Fluoride, Natrium Fluoride and Kalium Fluoride	LiF-NaF-KF	1, 5, 6
FLuZirK	A molten salt containing Kalium Fluoride and Zirconium Fluoride	$KF\text{-}ZrF_4$	1, 5
Hitec	A molten salt containing Natrium Nitrate, Natrium Nitrite and Kalium Nitrate	$NaNO_3\text{-}NaNO_2\text{-}KNO_3$	1, 5
Lithium fluoride	An inorganic compound used in molten salts	LiF	1, 7
NaFNaB	A molten salt containing Natrium Fluoride and Natrium Boron Fluoride	$NaF\text{-}NaBF_4$	1, 5
Plutonium trifluoride	An inorganic compound used in molten fuel salts	PuF_3	1, 7
Solar Salt	A molten salt containing Natrium Nitrate and Kalium Nitrate	$NaNO_3\text{-}KNO_3$	1, 5
Thorium tetrafluoride	An inorganic chemical compound used in molten fuel salts which reacts with moisture at high temperature	ThF_4	1, 7
Thorium-232	A primordial fertile isotope of Thorium	^{232}Th	1, 3, 4
Uranium tetrafluoride	An inorganic compound used in molten fuel salts	UF_4	1, 7
Uranium-233	A fissile isotope of Uranium with mass number 233	^{233}U	1, 3, 4

List of Units

Notation	Description	Symbol	Page List
Density ρ	SI unit of density	$[kg/m^3]$	1, 6, 8
Dynamic Viscosity μ	SI unit of kinematic viscosity	$[Pa \cdot s]$	1, 6, 8
Heat Capacity Cp	SI unit of amount of energy required to heat 1kg by 1degree	$[JKkg^{-1}]$	1, 8
Kinematic Viscosity ν	SI unit of kinematic viscosity	$[m^2s^{-1}]$	1
Mole n	SI unit of amount of substance	[mole]	1, 7
Pressure p	Non SI unit of pressure	[bar]	1, 8
Temperature T	Celsius, Non SI unit of temperature	[°C]	1
Temperature T	Kelvin, SI unit of temperature	[K]	1, 6, 8
Thermal Conductivity λ	SI unit of material property to conduct heat	$[Wm^{-1}K^{-1}]$	1, 8

Contents

1 Introduction

This chapter will describe how this thesis fits within the Master of Sustainable Energy Technology, abbreviated as SET, at Delft University of Technology. In this SET a student will obtain general knowledge of the Sustainable Energy solutions and chooses a track of interest in which to commence further research. Regarding the subject of this thesis, the track chosen is Nuclear Science and Engineering.

This chapter will give an introduction to nuclear energy research and how this thesis fits within the research framework. Since this paper is about Computational Fluid Dynamics with the use of the Lattice Boltzmann Method both these subjects will be introduced as well. Finally Palabos; the program used for this simulation, as well as the C++11 coding language will be discussed.

1.1. Sustainable Energy Technology

Not all energy is sustainable, and neither is all nuclear energy. The Cambridge dictionary defines sustainable energy as “Natural resources, environment energy that is produced using renewables, rather than using fuels such as oil or coal which cannot be replaced.”[7] This definition in the case of a Thorium fuelled Molten Salt Reactor (MSR) [30] is debatable. However since Thorium is much more abundant than commonly used nuclear fuels and since there are no emissions in the Thorium fuel cycle, Thorium is commonly accepted to be a sustainable energy source.

1.1.1. Nuclear Science and Engineering Profile

The SET Nuclear Science and Engineering profile focusses on green nuclear energy. That is why in the SET framework the focus lies on Molten Salt Reactors and the physical properties of the molten salts itself. Head of the profile is Prof. dr. ir. Jan Leen Kloosterman who is also the head of the research department NERA. SET student projects include the following.

- Numerical methods for safety analysis of MSR
- Experimental research into properties of liquid salts
- Numerical and experimental research into stability of salts
- Reactor physics analysis of advanced MSR designs

The subject of this thesis discusses the second bullet point.

1.2. Nuclear Energy and Radiation Applications

The Nuclear Energy and Radiation Applications (NERA) department, is located at Reactor Institute Delft (RID), and is part of the Department of Radiation Science and Technology of the Faculty of Applied Sciences at Delft University of Technology. The three major research disciplines within NERA are the following.[19]

- Nuclear Reactor Thermal Hydraulics
- Computational Methods in Reactor Physics
- Innovative Reactors and fuel cycles
- Chemistry of the fuel cycle

1.3. Change of Thesis Topic

The research on the physical properties of molten salts with use of the Lattice Boltzmann Method (LBM) at NERA, is led by Martin Rohde [25] [26] [27] [28] [29] , who is also the supervisor of this thesis. The lattice Bolztmann method will be introduced further on in Section 1.7.2. This thesis will analyse the simulation of the flow of Thorium fuel salt in a viscosity experiment. The experiment is led by Sara Mastromarino and this thesis was initially set up to provide a read-out graph on the sedimentation time of an object in an enclosure filled with Thorium fuel salt at high temperatures. Since the thorium fuel salt will be used in a Molten Salt Fast Reactor (MSFR), a short introduction of the reactor is required, after which the fuel salt shall be introduced in Section 1.5. The outline of the viscosity experiment is introduced in Section 1.6. Due to changes in the approach of this experiment the read-out graph is no longer necessary, however the code itself could be used for many other salt simulations of an object in a salt flow within an enclosed container, e.g. flow around the reactor core. The focus of this thesis has shifted towards the simulation of molten salt flow and its code written in C++ which shall be introduced in section 1.9 instead of the viscosity experiment because the Palabos library which is introduced in section 3.1 was difficult to read and documentation did not exist. This made it hard to complete the entire model within the allotted time.

1.4. Molten Salt Fast Reactor

The research on the MSFR is conducted in the framework of the Generation IV International Forum, abbreviated GIF. In the GIF framework two types of MSR reactors are researched: 1) One in which the fuel is dissolved in the salt and which operates in the fast neutron spectrum also known as a Molten Salt Fast Reactor (MSFR) and, 2) one in which a solid fuel is used and the salt only acts as a coolant typically referred to as a Fluoride salt cooled High-temperature Reactor (FHR) [11].

The European Union (EU), France and Russia are focusing on the development of a fast spectrum molten salt reactor capable of either breeding or the transmutation of actinides from spent nuclear fuel.[30] Hence the research done at NERA focusses on the MSFR and is conducted in the Safety Assessment of the Molten Salt Fast Reactor (SAMOFAR) framework.

The R&D targets for the MSFR formulated by GIF and presented in 2014 [11], are the following.

- Studying the salt chemical and thermodynamic properties, including transuranic elements
- Development of efficient techniques for gas extraction from the coolant
- System design: development of advanced neutronic and thermal-hydraulic coupling models
- Analysis of salt interactions with air or water in case of a severe accident
- Analysis of the accident scenarios (e.g. heat exchanger loss)
- Salt reprocessing: lanthanide and actinide reductive extraction tests

This thesis is part of the study on the thermodynamic properties of the salt, which falls under the first bullet point in the list above.

The MSFR operates, as stated earlier, through a coolant fuel combination. In other words, the fuel is contained within the coolant. Figure 1.1 shows a general example of an MSFR reactor. In general

the MSFR will require some ^{233}U (Uranium-233), within the salt containing ^{232}Th (Thorium-232), which will be elaborated on in Section 1.5.

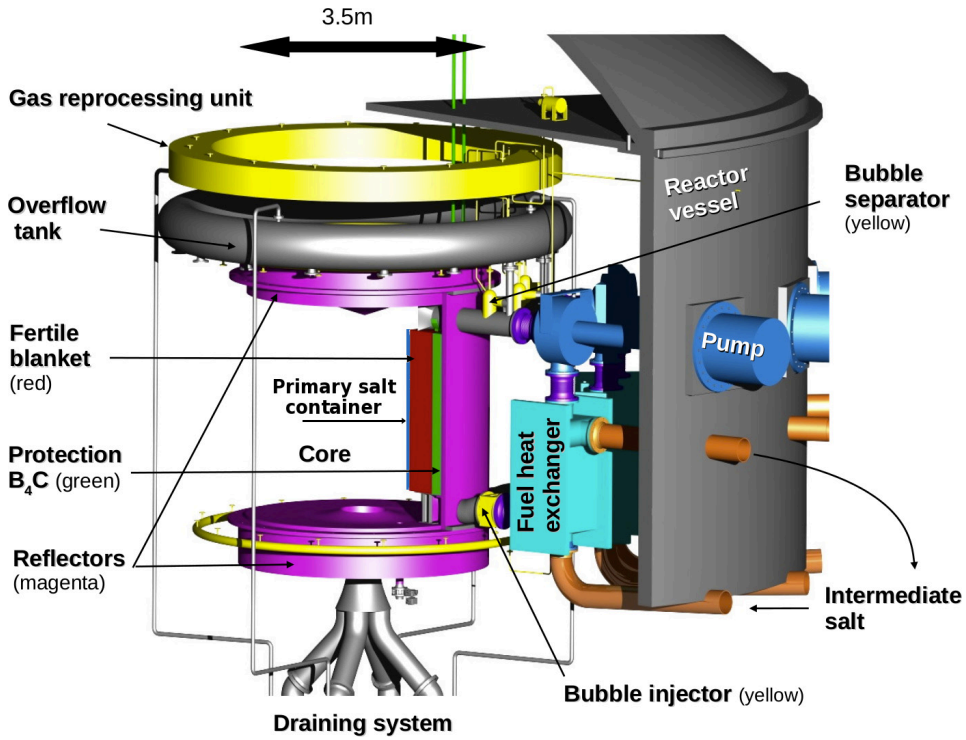


Figure 1.1: Example of a Molten Salt Fast Reactor Setup [6]

1.4.1. MSFR Components

To get a better understanding of the inner workings of the MSFR seen in figure 1.1 some of it's main components will discussed in the following.[6].

Reactor Core
Since the fuel salt exists everywhere in the primary cycle the definition of the core becomes the location within the MSFR where the most fission takes place.

Fuel Salt
The salt acts as coolant, moderator, and fuel. Section 1.5 will elaborate further on this.

Fertile Blanket
The fertile blanket is needed to improve the reactor's breeding capability, where breeding means the production of ^{233}U from neutrons colliding with ^{232}Th. Initial blanket salt does not contain any Uranium but does contain a higher percentage Thorium then the normal Fuel Salt composition.

Pyrochemical Reprocessing Unit
The fission products may contain unwanted soluble Lanthanides which need to be removed form the Fuel Salt. The pyrochemical reprocessing unit does this.

Gas Reprocessing Unit
The fission products can also contain insoluble unwanted elements which are extracted from the fuel salt by the Bubble Separator and are reprocessed in the Gas Reprocessing Unit. Since some required fission products are also insoluble, the Gas Reprocessing Unit will separate these and send them to the Bubble Injector.

Upper and Lower Reflectors
In order to shield the surrounding from the neutron flux, and to increase the fission probability, the upper and lower walls are covered with a neutron reflector. The side walls also contain a neutron reflector behind the fertile blanket increasing the breeding probability.

Neutronic Protection
In order to shield the pumps and the heat exchanger from the neutron flux a protection wall is added after the fertile blanket. However, placing the protection wall behind the side wall reflector may prove more efficient.

Heat Exchanger
Next to extracting heat for power generation the Heat Exchanger also allows control thermal control of the Reactor Core temperature by controlling the secondary salt flow rate and temperature.

Draining System
In case of unwanted operation parameters the draining system empties the reactor of Fuel Salt thereby stopping Fission from taking place. For safety reasons a frozen fuel salt plug is proposed, so that in case of a power failure the the plug will not be cooled anymore, melt, and subsequently empty the Reactor Core.

Now that the MSFR has been introduced, the following will focus on the Thorium fuel salt.

1.5. Thorium Fuel Salt

The use of a salt as a coolant over - for instance water - has several advantages.

- High heat capacity
- High boiling point
- No undesirable exothermal chemical reactions
- Ability to dissolve actinides
- Great insensitivity to radiation

The above mentioned points make a molten salt a very good option as coolant and to a lesser extend as a moderator for nuclear fission power plants.

The use of a molten salt however may prove to have some drawbacks, and therefore a closer look is taken on the current research on molten salts for modelling or for use in a reactor.

1.5.1. Types of Molten Salts used in Reactors

In the research on molten salt thermodynamic properties, three salt groups can be defined [31].

Chlorides

The Chloride group includes the following salts.

- KCl-$MgCl_2$ or short: CloKMag

Clokmag is currently studied regarding it's use in heat transfer piping but is not studied in the MSFR context.

Nitrates

The Nitrate group includes the following salts.

- $NaNO_3$-KNO_3 or by it's product name: Solar Salt
- $NaNO_3$-$NaNO_2$-KNO_3 or by it's product name: Hitec

The nitrates are generally used in the chemical industry and the solar salt is also used in the solar power plant as heat transporter. Again these salts are not researched for use in a MSFR.

Fluorides

The Fluoride group includes the following salts.

- 2LiF-BeF_2 or short: FLiBe
- LiF-NaF-KF or short: FLiNaK
- LiF-NaF-BeF_2 or short: FLiNaBe
- NaF-$NaBF_4$ or short: NaFNaB
- KF-ZrF_4 or short: FLuZirK

For the modelling of molten fuel salt, the fluoride group is researched, since these salts have a complex composition and have a high melting temperature it proves difficult to research their properties. Figure 1.2 shows the research results of various studies on the viscosity of FLiNaK. The research on the actual thorium fuel salt that is used in a MSR shall be presented in section 1.5.2.

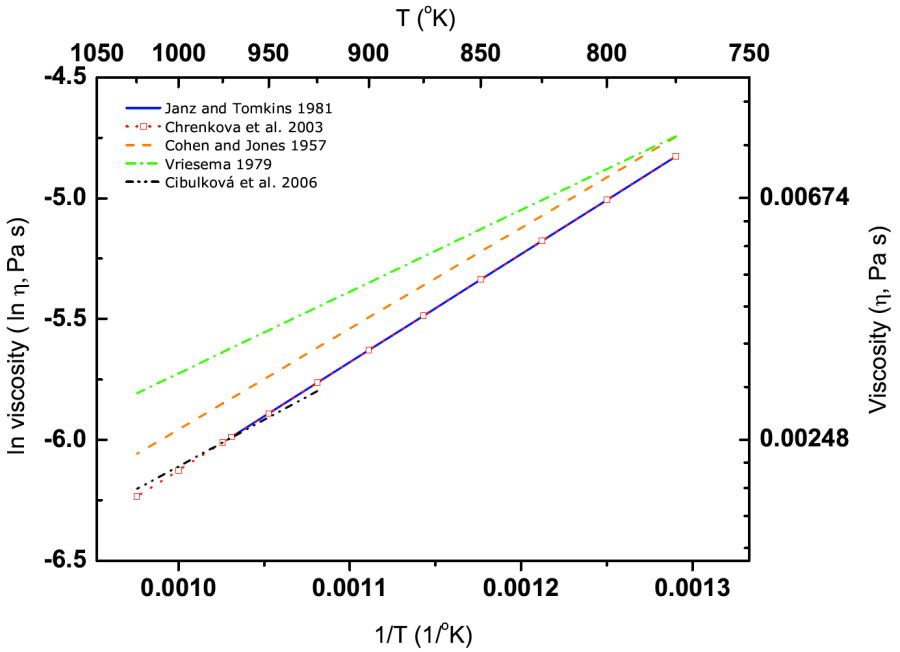


Figure 1.2: Research results of various studies on the viscosity of FLiNaK [31]

From figure 1.2 it becomes clear that there is currently no consensus on the viscosity of FLiNaK. Therefore more research is required.

In the research done by NERA, the salt studied is based on FLiNaK which does not have fissile elements in it and hence is not a fuel salt. Its thermodynamical properties are approximated by equation 1.1 and 1.2 [31].

$$\rho = 2579.3 - 0.6237 \cdot T \tag{1.1}$$

With Density ρ [kg/m^3] and Temperature T [K].

$$\mu = 0.0000249 \cdot 10^{\frac{1944}{T}} \tag{1.2}$$

With Dynamic Viscosity μ [$Pa \cdot s$] and Temperature T [K].

The actual fuel salt also has fissile components in it. How this effects the research is discussed next.

1.5.2. Properties of Molten Fuel Salt

The actual fuel salt contains Thorium, Uranium and Plutonium next to it's salt components. Two fuel salts were studied in the previous EU research projects financed by Euratom. The compositions of the salts that were studied are given by table 1.1.

Table 1.1: Composition of the Fuel Salts [2]

Fuel 1	LiF	ThF_4	UF_4	PuF_3
Composition [mole] %	78.6	12.9	3.5	5
Fuel 2	LiF	ThF_4	UF_4	$(TRU)F_3$
Composition [mole] %	77.5	6.6	12.3	3.6

Where LiF is Lithium fluoride, ThF_4 is Thorium tetrafluoride, UF_4 is Uranium tetrafluoride, PuF_3 is Plutonium trifluoride and TRU represents the transuranium elements which are mainly represented by PuF_3. [2].

The research on the properties of molten fuel salts are very important for the research on MSFR. It is necessary to describe the properties of molten fuel salts, as well as their importance, which will be done in the following.

Melting Point
The melting point should be as low as possible to achieve lower operating temperatures. This also reduces the risk of salts freezing inside the reactor.[2]

Vaporisation Pressure
The vaporisation pressure should be as low as possible since any vaporisation would change the composition of the molten fuel salt. [2]

Heat Capacity
The heat capacity should be as high as possible to allow for the maximum storage and transportation of heat.[2]

Thermal Conductivity
The thermal conductivity should be as high as possible to achieve an even spread of heat throughout the fuel salt.[2]

Density
The density needs to be known in order to determine the neutronic behaviour in the fuel salt.[2]

Viscosity
The viscosity should be known in order to have an idea about the fuel salt flow. Does it become turbulent at some place in the reactor, is there homogenous heat transfer or is there heat build up somewhere? [2]

Table 1.2 and 1.3 show the approximated thermodynamic properties of the fuel salts studied in the NERA SAMOFAR and GIF framework.

Table 1.2: Approximated Thermodynamic Properties of Fuel Salt 1 [2]

Property	Fuel 1	T range
Melting Point	872 [K]	
Boiling Point	2028 [K]	
Vapour Pressure	$\ln p = 14.263 - \frac{28469}{T}$ [bar]	800 - 2000 [K]
Heat Capacity	1020 $[JKg^{-1}]$	$800 - 2000$ [K]
Thermal Conductivity	1.7 ± 0.3 $[Wm^{-1}K^{-1}]$	1000 [K]
Density	$5155 - 0.8331 \cdot T$ $[kg/m^3]$	$800 - 1100$ [K]
Dynamic Viscosity μ	$\ln \mu \cdot 10^{-3} = 0.178 + \frac{403.5}{T-804.2}$ $[Pa \cdot s]$	$973 - 1273$ [K]

It must be stated that the Thermal Conductivity and Dynamic Viscosity in both tables are engineering estimates and should therefore be studied further.

Table 1.3: Approximated Thermodynamic Properties of Fuel Salt 2 [2]

Property	Fuel 2	T range
Melting Point	854 [K]	
Boiling Point	2015 [K]	
Vapour Pressure	$\ln p = 14.466 - \frac{28701}{T}$ [bar]	800 - 2000 [K]
Heat Capacity	1010 $[JKg^{-1}]$	$800 - 2000$ [K]
Thermal Conductivity	1.7 ± 0.3 $[Wm^{-1}K^{-1}]$	1000 [K]
Density	$5108 - 0.8234 \cdot T$ $[kg/m^3]$	$800 - 1100$ [K]
Dynamic Viscosity μ	$\ln \mu \cdot 10^{-3} = -0.48 + \frac{771.2}{T-765.2}$ $[Pa \cdot s]$	$973 - 1273$ [K]

The challenge in measuring the before mentioned properties of molten fuel salts has several causes which include the following.

- The high temperatures at which the measurements need to take place which is though on the measuring equipment
- The radioactivity of the fuel salt increases safety risks
- Reactions may take place with the container or measuring tools and may change the composition.
- Reactions may take place with the atmosphere and thereby may change the composition.

Keeping these challenges in mind, an experiment setup was proposed to determine the viscosity of the molten fuel salts. This experiment shall be discussed next.

1.6. Measurement of the Viscosity of Molten Salts

In order to determine the viscosity of molten Thorium fuel salt, an experiment is needed. Keeping the before mentioned challenges in mind, some constrictions to the experiment setup apply. The molten salt should be in an enclosed container and the material that the container is made of should be able to withstand high temperatures without deforming. In addition the container should not react with the salt and shield the surroundings from radiation.
The measuring equipment must be operated remotely and must also be able to withstand the high temperatures and - if they are in direct contact with the molten salt - should also not react with the salt. Finally, since resources are not infinite the proposed experiment setup must not be too expensive.

The following viscosity experiment setup was proposed by Martin Rohde as shown in figure 1.3.

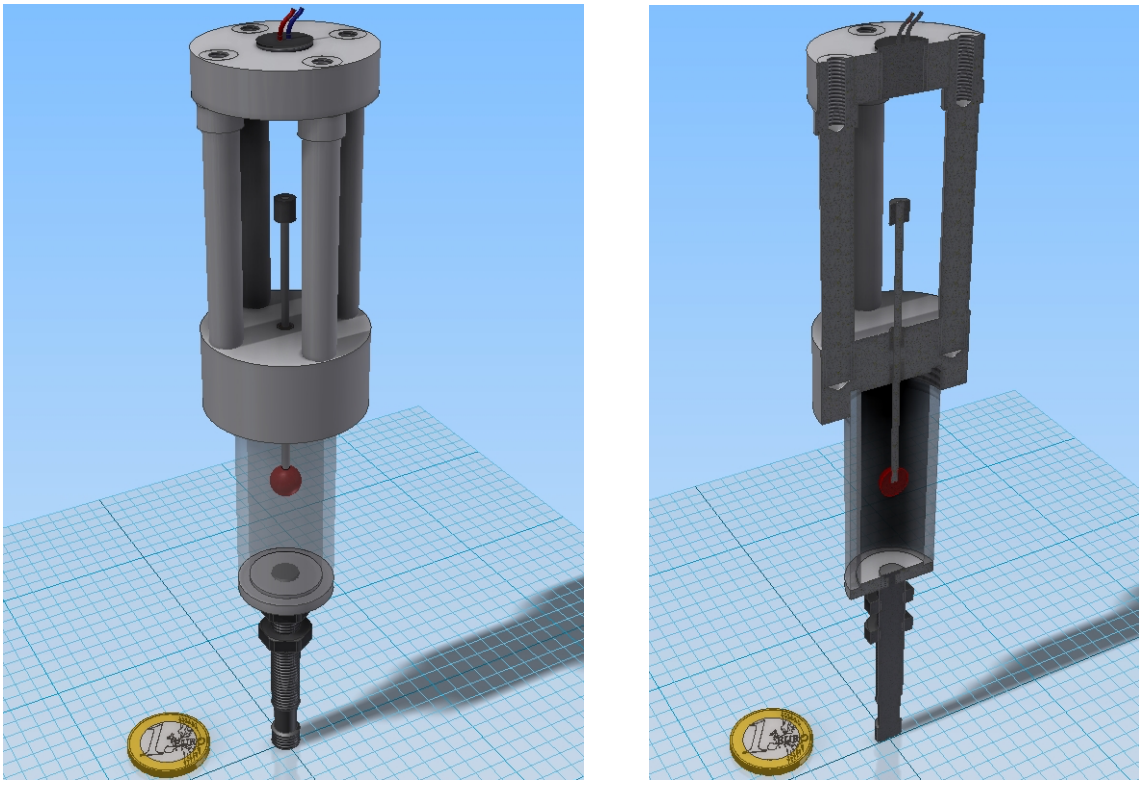


Figure 1.3: A settling sphere viscosity meter where the sphere in red travels through the molten salt. Courtesy of M.N. Verleg Design Engineer at Department of Electronic and Mechanical Development (Dienst Elektronische en Mechanische Ontwikkeling) (DEMO)

On the top of the viscosimeter an electromagnet is attached to hold the sphere, in red, in place at the top start position. The middle section of the device is cylindrical and contains the salt. On the bottom of the device a sensor is placed to detect the presence of the falling sphere. All dimensions are estimates, and for indication of the scale a euro coin is added to the drawing.

As a result of this test a sedimentation time at a range of operating temperatures is provided. Since the exact relation of the viscosity with temperature is unknown a simulation is required to get the equivalent dynamic viscosity. This read out graph can be generated from a Computational Fluid Dynamics (CFD) simulation. The following simulation is proposed.

1.6.1. Simulation

In earlier research the density was found within an operating temperature range of 800 to 1100 Kelvin as was shown by tables 1.2 and 1.3.[2] The density is related to the dynamic viscosity through the Reynolds number as shown by equation 1.3.

$$Re = \frac{\rho v L}{\mu} \tag{1.3}$$

Where ρ is the density of the molten fuel salt, v the characteristic velocity of the fuel salt, L the characteristic length scale and μ the dynamic viscosity.

For the simulation the Reynolds number is chosen as the main variable. This due to it being dimensionless and it defines the required dynamic viscosity. Since the density in equation 1.3 has a temperature range of 800 to 1100 Kelvin this range is also proposed for the simulation. Now if for each of the unique combinations of Reynolds number and temperature a simulation of the flow of the fluid around the settling sphere is carried out yielding, the velocity one could map the viscosity versus the sedimentation time and temperature. A curve fitting algorithm or, even better, a regression analysis would yield the relation of the dynamic viscosity with sedimentation time and temperature. But how can one simulate flow? The dynamics of a fluid are well described by the Navier-Stokes equations. Solving these is a complex and time consuming problem and for most engineering problems impossible to solve, hence it is best carried out by a computer. The scientific field of solving fluid flows is called Computational Fluid Dynamics (CFD). The software chosen for this simulation is called Palabos and is a open-source CFD LBM library written in C++. Lattice Boltzmann Method (LBM) is a part of the field of Computational Fluid Dynamics (CFD) and has an enormous advantage over conventional CFD methods because it is extremely parallel. The next section will explain in more detail what CFD and LBM are exactly. More information on Palabos will be given in Section 3.1 and more information on the C++ coding language in Section 1.9.

If the results for the density in earlier mentioned research would also be questioned, this simulation could easily be expanded to an NxNxN matrix with varying density and with varying Reynolds number versus a temperature range.
Although simulation time would increase significantly, this would probably yield the best result and a proper regression analysis is now required.

1.7. Computational Fluid Dynamics

As stated earlier, the Lattice Boltzmann Method (LBM) is a part of the field of Computational Fluid Dynamics (CFD). CFD aims to solve the Navier-Stokes equations. Equation 1.4 shows the Incompressible Navier-Stokes equation in the convective form.

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} - \nu \nabla^2 \mathbf{u} = \frac{1}{\rho}\nabla p + \mathbf{g} \tag{1.4}$$

Where on the left hand side the first term is the variation in time of fluid velocity, the second term is the convection term and the third the diffusion term. On the right hand side the first term is the internal source or specific thermodynamic work followed by the external source or work done on the fluid.
In equation 1.4 ν is the kinematic viscosity which is related to the dynamic viscosity through equation 1.5.

$$\nu = \frac{\mu}{\rho} \tag{1.5}$$

How should one solve the Navier-Stokes equation? Usually a problem is divided into multiple cells and these are solved cell by cell. However, does one consider to solve the equation per unit volume or unit mass? What if one cell relies on data of the next cell? For these problems numerous methods are created.

1.7.1. Conventional CFD Methods

Many methods exist in conventional CFD, and these methods will be discussed in the following.

Finite Volume Method

The Finite Volume Method (FVM) divides the domain in control volumes where the variables of interest are located in the centre of each control volume. Integrating over each control volume will eventually yield the desired variables.

$$\frac{\partial}{\partial t}\iiint \mathrm{Q}dV + \iint \mathrm{F}dA = 0 \tag{1.6}$$

Where Q is the vector of conserved variables, V the control Volume, F is the vector of the fluxes and A the surface area of the control volume. Some of the advantages of this method are low memory usage, high solution speed for large problems and the ability to simulate high Reynolds numbers. The main disadvantage is that complex shapes e.g. curves, require a lot more computing power. [21]

Finite Element Method

The Finite Element Method (FEM) with Galerkin's method of weighted residuals works by discretising a surface mesh into discrete elements. Integrating over each element will finally yield the desired variables.

$$R_i = \iiint W_i Q dV^e \tag{1.7}$$

Where R is the residual on vertex i, W is the weight factor, V^e the Element volume.
The main advantage is that FEM is much more stable then FVM but this comes at the cost of speed and memory usage which are also the two biggest disadvantages. [13]

Finite Difference Method

The Finite Difference Method (FDM) works by approximating the partial derivates in the differential equation with linear functions on certain grid points.

$$\frac{\partial \mathrm{Q}}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} \tag{1.8}$$

Where Q is the vector of conserved variables and F, G and H the fluxes in the x, y and z direction respectively.
The main advantage is the easy implementation. Drawbacks include problems along curved boundaries, difficult stability, and mesh adaptation.[24]

Finally, one other well known CFD method is the Lattice Boltzmann Method (LBM) which has the main advantage, as stated earlier, that it is highly parallel which makes it extremely fast with modern day computers or a video card solver. The main disadvantage is that it is only valid for very low Mach numbers. In other words the compressibility error takes over relatively quickly. In the case of this thesis this is not a problem since thorium salt is very viscous therefore this method is chosen. Section 1.7.2 will explain the method in greater detail.

1.7.2. Lattice Boltzmann Method

The statistics describing the kinetics of particles in an ideal gas was first formulated by Ludwig Eduard Boltzmann [4] who was a Austrian Physicist together with James Clerk Maxwell they formulated the Maxwell-Boltzmann distribution.

Figure 1.4: Ludwig Eduard Boltzmann 1844-1906 Austrian Empire

Boltzmann's General Principle
Ludwig Boltzmann described the speed of freely moving particles that don't interact with each other except for a collision term where they interchange momentum. Equation 1.9 shows his theory.

$$\frac{\partial f}{\partial t} = \left(\frac{\partial f}{\partial t}\right)_{\text{force}} + \left(\frac{\partial f}{\partial t}\right)_{\text{diff}} + \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \qquad (1.9)$$

Where f is the Maxwell-Boltzmann particle distribution with it's partial derivatives divided over a term of all forces acting on the particles, a diffusion term and a collision term.

From probability distribution to LBM
In the late 1940's John von Neumann was working on one of the first computers. His goal was to create a model of the human brain in order to build a machine that can solve very complex problems. At the same time Stanislaw Ulam was working on crystal growth using a simple lattice model. Neumann with the help of Ulam stated that processors and the data - instead of top down - should be on the same level. They proposed to divide a complex problem like the brain to be divided up in cells, each with their own processor, to be able to determine their internal state. This model is called the Cellular Automata (CA). [9]

It was still extremely difficult to implement von Neumann's rules in a computer model. In the 1970's Hardy, Pomeau, and de Pazzis created a lattice gas model based on Boltzmann's principle. This model was in fact a form of CA which formed the name of the new model namely the Lattice Gas Automata (LGA). This model consisted of fully discrete dynamics of particles colliding. However some disadvantages became clear, namely: the rise of statistical noise which needed averaging hence increasing computational cost. [9]

The Lattice Boltzmann Method (LBM) originates from the Lattice Gas Automata (LGA), the difference between them is that LGA is based on Boltzmann's principle and the LBM implements it directly, and in the 1980's McNamara and Zanetti where one of the first to do so. They proposed to extend the boolean dynamics of the automation with real numbers representing the probability of a cell being in a certain state thereby neglecting many body correlation states. The big advantage of LBM was that they eliminated the statistical noise and thereby decreased the computational cost. [9]

A lattice is divided in cells. Hence for a 3 dimensional problem the lattice would consist of NxNxN cells. Each cell in the lattice has a particle population and the directions of particle movement as can be seen in figure 1.5.

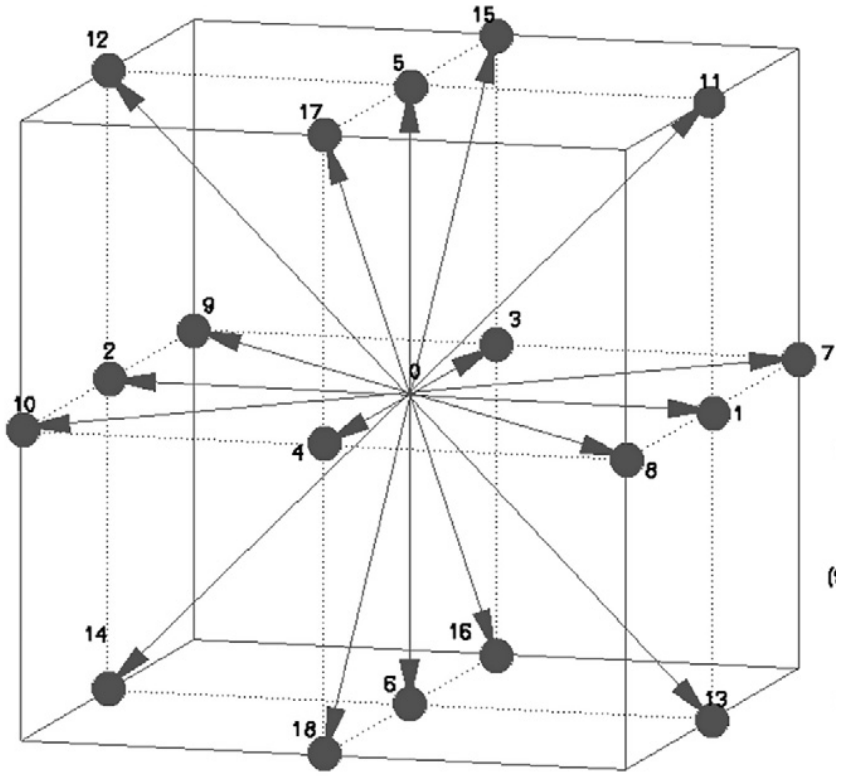


Figure 1.5: A cell in a 3 Dimensional Lattice with 19 movement directions

In order to solve the desired variables, a certain amount of directions is necessary. In the case of this thesis where only the forces acting on the falling object inside the container or viscosimeter are of interest, thereby taking into account the 3 dimensions, it can be shown that 19 directions will suffice [8].

In the case of an infinite lattice with no object in the flow, the standard Lattice Boltzmann Equation (LBE) would suffice, when a boundary exists however, the method needs an additional theory dealing with the fluid - boundary interaction. The chapter on theory chapter 2 will explain this in further detail.

1.8. Palabos

In order to implement LBM on a computer, naturally, software is required, but where there are many readily available programs - both commercial and open-source for conventional CFD - this not the case for LBM.

Most of the available software on LBM is commercially licensed and very expensive. Only a few open-source programs or libraries exist and most of them are based the Open LB code project or Open-source Lattice Boltzmann code. [33] Palabos is such a library and was chosen because of it's size and most up to date code collaborations. Palabos is a joint project between academia (University of Geneva) and commercial businesses (FlowKit). Palabos is written in C++ hence in next section this language will be introduced. [16]

1.9. C++

In 1979 Bjarne Stroustrup, a Danish computer scientist, wrote his Phd Thesis titled "C with Classes". He set out to combine the low-level memory control of the C language with the ease of use of the language Simula.

In 1983 the name was changed from "C with Classes" to "C++" which instantaneously shows how Bjarne thought of the C language since the ++ in coding means an increment of the variable. The most notable differences where virtual functions, overloading, referencing with ampersand and const qualifier.

In 1990 the ANSI C++ committee was founded and modern C++ was born and standardised. Following are some of the major updates since then, and only the major updates are shown for conveniance.

- C++98: added dynamic casting, mutable, bool, templates, string and iostream.
- C++03: was a minor revision added value initialisation
- C++11: was a major update added delegating, inheritance, nullptr, lambda expressions and many more
- C++14: was a minor revision added std::makeunique and some other library changes

In this thesis the C++11 language will be used since this was the last major update and this is the language version taught at Delft University of Technology by Matthias Möller who is member of this thesis committee.

2

Theory

As stated in the introduction, LBM is a different approach to solving the Incompressible Navier Stokes equations. Since the computer model of the viscosity experiment discussed in this thesis only deals with 3D problems, the theory discussed will be limited to 3D problems.

In CFD, normally the vector notation is used to reduce equation size and a set of partial differential equations are solved using a complex solver in each of the cartesian directions.
LBM has a very different approach. In LBM the moments of the particle population distribution are used to approximate a solution to the set of Incompressible Navier Stokes Equations. In addition to that, LBM uses the index notation instead of the vector notation.

In order to understand the bigger picture first, an explanation as to what the lattice is and why certain directions are chosen, will be provided. Next, the parameters used in LBM are explained followed by the governing equations used in the LBM model of this thesis. Finally, the Boundary Conditions will be explained and a detailed description will be given of the de Boundary Condition used in the model.

2.1. Lattice

LBM uses a so called lattice, let this lattice be a set of N_x by N_y, N_z cubic cells. Each cell has a particle population described by the Boltzmann distribution. If one is to describe the interaction between cells, a number of directions should be chosen. Each direction holds q set of variables in $i = 0 \cdots q - 1$ for carrying this data to the neighbouring cell. [17]

2.1.1. Directions

The first direction is $i = 0$, namely the current cell itself. If only the neighbours sharing a plane would be considered, a set of 6 directions should be added resulting in 6+1=7 sets of variables. If one is to include the neighbours only sharing a side, this increases to 7+12 = 19. Finally, if also its neighbours who only share a corner are included, this would result in 19+8=27 indices.

It can be shown that in a 3 dimensional problem, a set of a minimum of 19 indices is required to solve the forces properly. [8] The derivation and evidence for this statement is far outside the scope of this thesis. For now, it is enough to state that the notation used is denoted as D3Q19 or Number of Dimensions 3 number of sets of q variables 19.

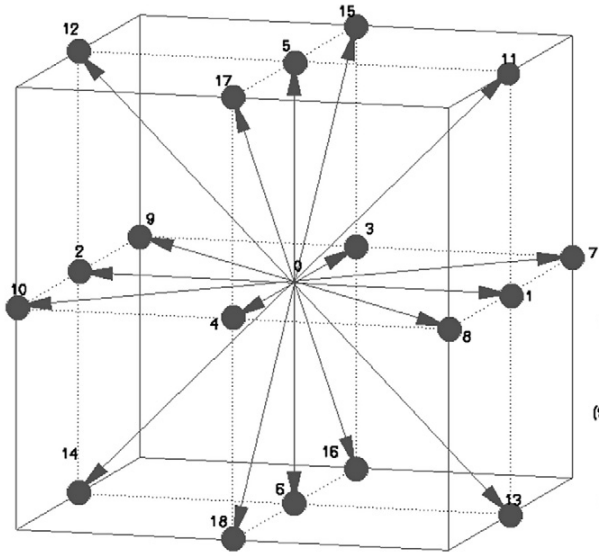


Figure 2.1: A cell in a 3 Dimensional Lattice with 19 movement directions

Figure 2.1 shows the general layout of the cells forming the lattice. As shown there are 19 sets of q variables. These directions indicate the number of discrete velocities $\mathbf{c}_i$ and are given by equation 2.1 [22].

$$\mathbf{c}_i = \begin{cases} (0,0,0) & i = 0; \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & i = 1,\dots,6 \\ (\pm 1,\pm 1,0),(\pm 1,0,\pm 1),(0,\pm 1,\pm 1) & i = 7,\dots,18 \end{cases} \tag{2.1}$$

The lattice weights for the D3Q19 model are given by equation 2.2 [22].

$$\mathbf{w}_i = \begin{cases} 2/9 & i = 0; \\ 1/18 & i = 1,\dots,6 \\ 1/36 & i = 7,\dots,18 \end{cases} \tag{2.2}$$

An explanation of how these lattice weights are used will be given in section 2.2.3. Now that a lattice consisting of a number of cells has been defined, the question remains as to how one should choose the dimensions of each cell, and what kind of implications this may have for the Compressibility error.

2.1.2. Lattice Parameters and Limitations

In LBM all physical variables are made dimensionless and discretised afterwards. As an example let's consider the cell spacing ls_p which in physical units must be dimensionless. One can do this with the use of the characteristic length scale $l_{0,p}$ as can be seen in equation 2.3. The characteristic length scale can be seen as the smallest length of any object in the simulation.

$$ls_d = \frac{ls_p}{l_{0,p}} \tag{2.3}$$

It can be stated that ls_d, the dimensionless space interval, and lt_d, the dimensionless time interval, are in unity, meaning both their value is 1 by definition. [18] ls_d must now be discretised. This can be done as shown in equation 2.4.

$$\delta_x = \frac{ls_d}{N} = \frac{1}{N} \tag{2.4}$$

Where the discrete space interval δ_x is defined as the reference length in dimensionless units divided by N, the number of cells in the same direction as the characteristic length scale. [18]
lt_d must also be discretised and can be done so in a similar manner as can be seen in equation 2.5.

$$\delta_t = \frac{lt_d}{N_{\text{iter}}} = \frac{1}{N_{\text{iter}}} \tag{2.5}$$

Where the discrete time interval δ_t is defined as the reference time in dimensionless units divided by the number of iterations needed to reach this time. [18]

All other variables can now be directly converted from the dimensionless system through dimensional analysis. However, the question remains as to how one should choose the time interval and space interval to begin with?

In LBM the speed of sound is given by equation 2.6.

$$c_s = \frac{1}{\sqrt{3}} \tag{2.6}$$

Since LBM does not support supersonic flow, the first limitation is that the characteristic velocity $u_{0,lb}$ should not be greater then the local speed of sound c_s. This leads to the following constraint as shown by equation 2.7 which ensures that since the local speed of sound is set the model will keep well within the incompressible region.[18]

$$\delta_t < \frac{\delta_x}{\sqrt{3}} \tag{2.7}$$

Another limitation is that the compressibility error, which is of the order of $\epsilon(Ma) \sim \delta_t^2/\delta_x^2$, should be minimised. A limitation which is similar to this compressibility error is the lattice error or in other words: the error in recovering the macroscopic variables. Since LBM is second order accurate, the lattice error is of the order $\epsilon(\delta_x) \sim \delta_x^2$. Both errors should be minimised, it is therefore good practice to keep the compressibility error of the same order as the lattice error $\epsilon(Ma) \sim \epsilon(\delta_x)$. This leads to the second constraint shown by equation 2.8. [18]

$$\delta_t \sim \delta_x^2 \tag{2.8}$$

It must be stated that δ_x and δ_t will be used to convert the discrete variables back to physical units. In order to do so those unknown variables must first be calculated using a set of equations which are explained in the next section.

2.2. Governing Equations

In order to understand the theory behind the model, the governing equations must be discussed. This section will attempt to give more insight into the theory by explaining the equations that form the basis of the model, starting with the dynamics of a settling sphere followed by the equations involved with the LBM.

2.2.1. Settling Sphere

As mentioned in the introduction, the proposed viscosity experiment was a settling sphere model. In order to describe the movement of the settling sphere one needs the equations of motion and the drag force acting on the sphere.

Equations of motion

For 3D movement one starts with Newton-Euler's law about it's centre of mass.

$$\begin{pmatrix} \mathbf{F} \\ \mathbf{T} \end{pmatrix} = \begin{pmatrix} m\mathbf{I}_3 & 0 \\ 0 & \mathbf{I}_{\text{cm}} \end{pmatrix} \begin{pmatrix} \mathbf{a}_{\text{cm}} \\ \boldsymbol{\alpha} \end{pmatrix} \begin{pmatrix} 0 \\ \boldsymbol{\omega} \times \mathbf{I}_{\text{cm}}\boldsymbol{\omega} \end{pmatrix} \tag{2.9}$$

Where $\mathbf{F}$ is the force acting on the body, $\mathbf{T}$ is the torque acting on the body, m is it's mass, $\mathbf{I}_{cm}$ it moment of inertia about the centre of mass, $\mathbf{a}_{cm}$ the planar acceleration, $\boldsymbol{\alpha}$ the rotational acceleration and $\boldsymbol{\omega}$ it's rotational velocity.

First the mass shall be calculated from it's volume and density. Since the sphere in the algorithm is not an exact sphere it's volume is not simply $4\pi r^3/3$. The sphere used in the algorithm consist of a set of tetrahedrons as can be seen in figure 2.2

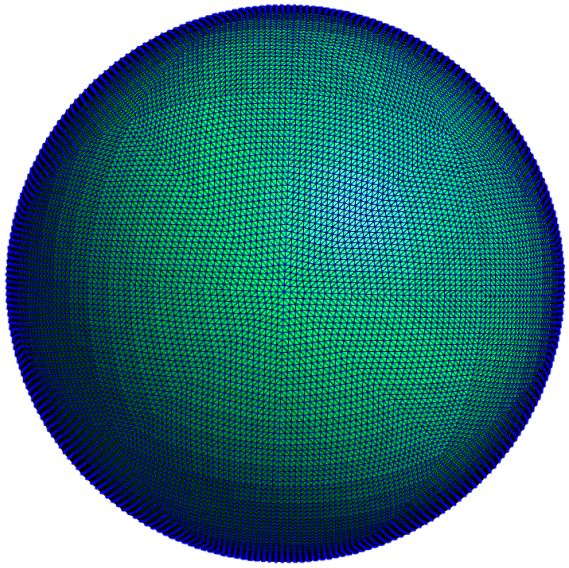


Figure 2.2: A sphere consisting of 49152 tetrahedrons

In order to calculate it's mass, first the centre of mass needs to be calculated. Assuming a homogenous density and constant gravitational field, the centre of mass coincides with the centroid and can be calculated with equation 2.10

$$\mathbf{cm} = \frac{\sum_{i=1}^{n} \mathbf{r}_i}{n} \tag{2.10}$$

The mass can then be calculated with 2.11 where the centre of mass is the top of the inward facing tetrahedron.

$$m = \rho_s V = \rho \sum_{i=1}^{n} \frac{|(\mathbf{a}_i - \mathbf{d}_i) \cdot ((\mathbf{b}_i - \mathbf{d}_i) \times (\mathbf{c}_i - \mathbf{cm}))|}{6} \tag{2.11}$$

Where ρ_s is its density, V it's volume which is obtained by taking the sum of the volume of all the tetrahedrons and $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{cm}$ are the Cartesian coordinates of the vertices of each tetrahedron.

Now that the mass is known, the moment of inertia needs to be determined. Again, since it's not a perfect sphere, one cannot use $I = 2/5mr^2$ and hence some adjustments need to made as can be seen in equation 2.12.

$$\mathbf{I} = \sum_{i=1}^{n} m_i \left((\mathbf{r}_i \cdot \mathbf{r}_i) \mathbf{E} - \mathbf{r}_i \otimes \mathbf{r}_i \right) \tag{2.12}$$

Where $\mathbf{E}$ is the identity tensor given by equation 2.13.

$$\mathbf{E} = \mathbf{e}_1 \otimes \mathbf{e}_1 + \mathbf{e}_2 \otimes \mathbf{e}_2 + \mathbf{e}_3 \otimes \mathbf{e}_3 \tag{2.13}$$

Resulting in the moment of inertia matrix as shown by equation 2.14.

$$\mathbf{I} = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{2.14}$$

Where $I_{yx} = I_{xy}$, $I_{zx} = I_{xz}$ and $I_{zy} = I_{yz}$. Now that the equations of motion are complete one needs to consider the forces acting on the body.

Drag
The drag of a settling sphere can be divided into three regions. In the region where the Reynolds number is sufficiently small: $Re < 1$ Stokes law holds, as shown by equation 2.15.

$$w = \frac{2\left(\rho_s - \rho_f\right) g r^2}{9\mu} \tag{2.15}$$

In the region where the Reynolds number is intermediate: $1 < Re < 1000$. an empirical solution is required as presented by Schiller and Naumann and is shown by equation 2.16.

$$\frac{F}{\rho_f U^2 A} = \frac{12}{Re}\left(1 + 0.15 Re^{0.687}\right) \tag{2.16}$$

Finally, for higher Reynolds numbers: $Re > 1000$, Newtonian drag holds and is shown by equation 2.17.

$$C_d = \frac{F_d}{0.5\rho_f U^2 A} \approx 0.44 \tag{2.17}$$

Since all three regions assume an infinite fluid in all directions it would be extremely difficult to use them in this model since there are walls on all sides and they interact with the fluid. Therefore a more complex method must be used.

2.2.2. Lattice Boltzmann Method

First it must be stated that the following theory is obtained from the thesis “Extending The Lattice-Boltzmann Method” by M. Rohde 2004 [25] unless stated otherwise. However a slightly different notation is used.

The most important equation is the Lattice Boltzmann Equation, abbreviated LBE. In computer algorithms this equation is split in an collision step as shown by equation 2.18 and a streaming step as shown by equation 2.19.

The collision step, equation 2.18, describes the fictitious distribution interaction between cells.

$$f_i'\left(\mathbf{r}, t + \delta t\right) = f_i\left(\mathbf{r}, t\right) + \Omega_i^{\mathrm{BGK}} f\left(\mathbf{r}, t\right) \tag{2.18}$$

Where $f_i'\left(\mathbf{r}, t + \delta t\right)$ is the term describing the distribution after collision, with the particle population denoted by f_i' which is the Maxwell-Boltzmann distribution in the i-th direction at location $\mathbf{r}$ at time $t + \delta t$. $f_i\left(\mathbf{r}, t\right)$ represents the particle population in the i-th direction at location $\mathbf{r}$ and time t before the collision step takes place.
Finally the actual collision is denoted by $\Omega_i^{\mathrm{BGK}} f\left(\mathbf{r}, t\right)$ which is the Bhatnagar Gross Krook Operator which will be discussed in detail further on in section 2.2.3.

The streaming step describes the kinetics of particles travelling outward or remaining in the cell as shown in equation 2.19.

$$f_i\left(\mathbf{r} + \mathbf{c}_i \delta t, t + \delta t\right) = f_i'\left(\mathbf{r}, t + \delta t\right) \tag{2.19}$$

Where $f_i\left(\mathbf{r} + \mathbf{c}_i \delta t, t + \delta t\right)$ represents the transport part, with f_i the particle population in the i-th direction at location $\mathbf{r}$ transported to location $\mathbf{r} + \delta_x$ over a time interval $t + \delta t$ with discrete velocity c_i.
The right hand side denoted by $f_i'\left(\mathbf{r}, t + \delta t\right)$ then becomes the new distribution of particles after transportation or streaming has taken place.

Macroscopic variables can be obtained by taking the moments of the particle distribution for each cell. In order to do so, a transformation matrix is required to project the particle distribution in velocity space onto the macroscopic variables in the moment space which shall be further explained in section 2.2.3

The density is the first moment of the particle distribution as shown by equation 2.20 [20]

$$\rho = \sum_{i=0}^{q-1} f_i^{eq}\left(\mathbf{r}, t\right) \tag{2.20}$$

Where the density ρ is the sum of the equilibrium particle distributions f_i^{eq} in all i directions at location $\mathbf{r}$ and time t.

The second moment of the equilibrium particle distribution is shown by equation 2.21 and represents the momentum.[20]

$$\rho\mathbf{u} = \sum_{i=0}^{q-1} \mathbf{c}_i f_i^{eq}(\mathbf{r}, t) \tag{2.21}$$

Where $\rho\mathbf{u}$ is the momentum at location $\mathbf{r}$ and time t which is obtained by taking the sum in all i directions of the discrete velocity $\mathbf{c}_i$ multiplied by the equilibrium particle population f_i^{eq} at location $\mathbf{r}$ at time t.

The conservation laws follow from the moments from the non-equilibrium particle distributions. The law for conservation of mass is given by the first moment of the non-equilibrium particle distribution as shown by equation 2.22.[20]

$$0 = \sum_{i=0}^{q-1} f_i(\mathbf{r}, t) \tag{2.22}$$

Equation 2.22 states that the sum of the non-equilibrium particle distributions f_i in all i directions at location $\mathbf{r}$ and time t should be zero.

The law for conservation of momentum is given by the second moment of the non-equilibrium particle distribution as shown by equation 2.23.[20]

$$0 = \sum_{i=0}^{q-1} \mathbf{c}_i f_i(\mathbf{r}, t) \tag{2.23}$$

Equation 2.23 states that the sum in all i directions of the discrete velocity $\mathbf{c}_i$ multiplied by the non-equilibrium particle population f_i at location $\mathbf{r}$ at time t should be zero.

The third moment represents the kinetic energy but is left out for simplicity. These moments are obtained by the so called Chapman-Enskog analysis which involves a Taylor series expansion of the particle distribution. This analysis also verifies the amount of directions needed in order to approximate variables within a certain error range. If for instance higher order moments are required to describe a more complex model, more directions could be needed, but such an analysis is outside the scope of this thesis.[20]

2.2.3. Bhatnagar Gross Krook Operator

The Bhatnagar Gross Krook Operator [3], abbreviated BGK, is a simple Single Relaxation Time (SRT) collision operator. Its main advantage over the old collision integral and the newer Multiple Relaxation Time (MRT) is its simplicity and therefore shorter computation time. On the other hand it must be said that MRT can simulate much higher Reynolds numbers then SRT.

Equation 2.24 shows the BGK operator.

$$\Omega_i^{\mathrm{BGK}} = -\omega\left(f_i - f_i^{\mathrm{eq}}\right) \tag{2.24}$$

Where ω is the relaxation frequency as shown by equation 2.26 and f_i^{eq} is the equilibrium solution to the particle distribution function given by equation 2.25.[12]

$$f_i^{\mathrm{eq}} = \mathbf{w}_i\left[\rho + \rho_0\left(\frac{\mathbf{c}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{c}_i \cdot \mathbf{u})^2}{2c_s^4} - \frac{\mathbf{u}}{2c_s^2}\right)\right] \tag{2.25}$$

In equation 2.25 $\mathbf{w}_i$ is the weight factor that was shown by equation 2.2, ρ is the density $\mathbf{c}_i$ the lattice discrete velocities there were presented by equation 2.1, c_s the speed of sound that was given by

equation 2.6 and $\mathbf{u}$ the velocity.

The relaxation frequency is directly related to the kinematic viscosity and controls how fast equilibrium is approached.[22]

$$\omega^{-1} = \tau = 3\nu + 0.5 \tag{2.26}$$

The kinematic viscosity is on the other hand is directly related to the Reynolds number as shown in equation 2.27.

$$Re = \frac{u_{0,\mathrm{LB}} N \delta_x}{\nu} \tag{2.27}$$

The macroscopic variables could be obtained by taking the moment of the particle distribution. For the D3Q19 model the macroscopic variables are shown by equation 2.28

$$m_i = \left[\rho\ e\ \epsilon\ j_x\ q_x\ j_y\ q_y\ j_z\ q_z\ 3p_{xx}\ 3\pi_{xx}\ p_{ww}\ \pi_{ww}\ p_{xy}\ p_{yz}\ p_{zx}\ t_x\ t_y\ t_z\right]^T \tag{2.28}$$

Where ρ is the density, e is the energy, ϵ is related to the square of the energy, j is the mass flux, q the energy flux, p the viscous stress tensor and π the momentum flux tensor. [22]

Now let's consider a transformation matrix M such that each of the moments of the macroscopic variables, shown by equation 2.28 and can be obtained from the particle population $m_i = Mf_i$. This transformation matrix is given by equation 2.29.

$$M = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
-30 & -11 & -11 & -11 & -11 & -11 & -11 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\
12 & -4 & -4 & -4 & -4 & -4 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & -4 & 4 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & -4 & 4 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
0 & 2 & 2 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 \\
0 & -4 & -4 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -2 & -2 & -2 & -2 \\
0 & 0 & 0 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -2 & -2 & 2 & 2 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1
\end{bmatrix} \tag{2.29}$$

The macroscopic variables can then be obtained by substituting 2.29 into equation 2.24 resulting in equation 2.30.

$$\Omega_i^{\mathrm{BGK}} = -M^{-1}\omega\left(m_i - m_i^{\mathrm{eq}}\right) \tag{2.30}$$

Where M^{-1} is the inverse transformation matrix, ω the relaxation frequency and $m_i - m_i^{\mathrm{eq}}$ the non-equilibrium and equilibrium moments in the i-th direction.

Substituting equation 2.30 into equation 2.18 results in the complete collision step with the BGK and can now be seen as the Lattice Boltzmann Bhatnagar Gross Krook Method, abbreviated as LBGK as

shown by equation 2.31.

$$f_i'(\mathbf{r}, t+\delta t) = f_i(\mathbf{r}, t) - M^{-1}\omega\left(m_i - m_i^{\mathrm{eq}}\right) f(\mathbf{r}, t) \tag{2.31}$$

If a lattice would be without an object in the flow and would be infinite in each direction this would be sufficient, but since this would not be advantageous in a settling sphere viscosity experiment, one needs to consider boundary conditions.

2.3. Boundary Conditions

When a fluid node is next to a wall and a streaming step takes place, an interaction will ensue with this wall. However, since the wall in this case is a solid, and the LBGK only works for fluids, extra modelling is required. In general. two different bounce back approaches exist for both the on-lattice case and the off-lattice case.

2.3.1. Bounce Back

Halfway bounce back and full way bounce back can be slightly confusing. One could also speak of half-way bounce back with either full or partial replacement. Hence in the halfway bounce back only the unknown particle populations are replaced with the direct opposite particle population and in the full-way bounce back all particle populations are replaced.

Full Bounce Back

Full Bounce Back can be seen as a local fluid node near a wall where all incoming particles are reflected back in the opposite direction. This step completes the collision step for this node. It is quite clear that this very simple approach is very fast but also the most inaccurate. Full bounce back as stated earlier is a dubious name since the wall is still half way between the fluid and solid node.

Halfway Bounce Back

In Halfway bounce back the wall is also halfway between a fluid node and a solid node as can be seen in figure 2.3. However, now only the populations that are unknown after the collision step are replace by the population in the opposite direction. This step now completes the streaming step.

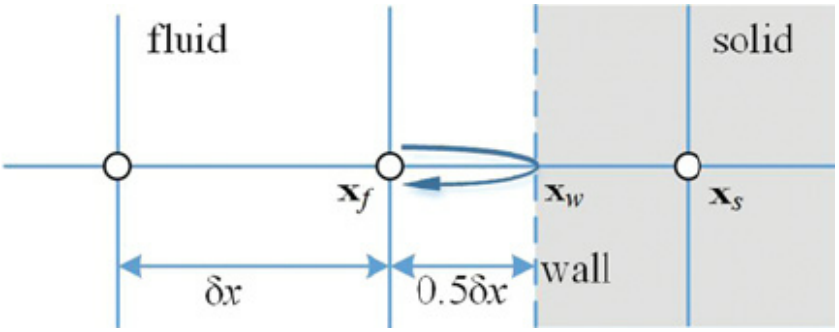


Figure 2.3: Halfway Bounce Back

One could say that the Full Way Bounce-Back is modifying the collision step and the Half Way Bounce Back is modifying the streaming step. [1]

2.3.2. Wet Node

A different approach is considering the wall as a fluid with a very high density, but it has been shown that this approach leads to an unphysical high jump in pressure near the wall and is therefore not considered for this model.

2.3.3. On-Lattice

On-lattice can be explained as the wall surface directly coinciding with the lattice. It is a very simple but useful boundary method especially in 2D simulation with straight walls. On the other hand, a sphere would in that case by step wise approximation of a sphere. It is obvious that this is not something that this is inaccurate, and hence, a different approach is needed.

2.3.4. Off-Lattice

In the off-lattice approach interpolation is used to calculate an area of volume that does not coincide with the lattice. There is a wide variety of proposals on how this should be done, but since the model was made in Palabos, the selection of models were limited to those included in the library. Therefore the following combined model shall be used.

2.3.5. Off-Lattice Moving Wall

Since the Palabos library doesn't provide a scheme for the moving wall no-slip method volumetric correction as proposed by M. Rohde [25], a different approach is taken. The boundary condition applied in the model is the Off-Lattice Guo extrapolation [12] method chosen for convenience and the movement is then done by an Inamuro iteration. [14]

Guo Extrapolation

In this method the streaming step remains unchanged hence this is a full bounce back approach. The method is an off-lattice approach as can be seen in figure 2.4.

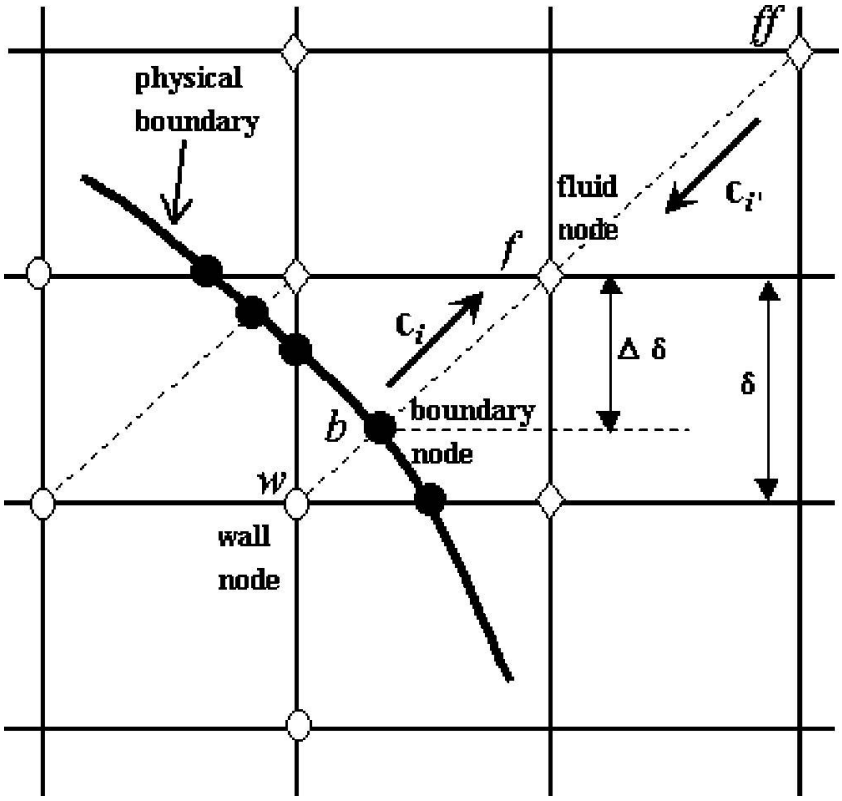


Figure 2.4: Guo Extrapolation

From figure 2.4 it becomes clear that $\mathbf{r}_w$ and $\mathbf{r}_f$ intersect at $\mathbf{r}_b$ where b denotes the boundary. The length of Δ which is the fraction in the fluid is given by equation 2.32.[12]

$$\Delta = \frac{|\mathbf{r}_f - \mathbf{r}_b|}{|\mathbf{r}_f - \mathbf{r}_w|} \tag{2.32}$$

The post collision particle distribution, $f'(\mathbf{r}, t + \delta_t)$ is split in an equilibrium, and non-equilibrium part. The equilibrium part is then approximated with a fictitious one first as shown by equation 2.33. [12]

$$\bar{f}_i^{\text{eq}}(\mathbf{r}_w, t + \delta_t) = \omega_i \left[\bar{\rho}_w + \rho_0 \left(\frac{\mathbf{c}_i \bar{\mathbf{u}}_w}{c_s^2} + \frac{(\mathbf{c}_i \bar{\mathbf{u}}_w)^2}{2c_s^4} - \frac{\bar{\mathbf{u}}_w^2}{2c_s^2} \right) \right] \tag{2.33}$$

Where $\bar{\rho}_w$ is approximated through 2.36 and $\bar{\mathbf{u}}_w$ can be determined through extrapolation using either equation 2.34 for $\Delta\delta_x \geq 0.75\delta_x$.

$$\bar{\mathbf{u}}_{w1} = \frac{\left(\mathbf{u}_b + (\Delta - 1)\,\mathbf{u}_f\right)}{\Delta} \tag{2.34}$$

Where $\mathbf{u}_f = \mathbf{u}(\mathbf{r}_f)$, the velocity at the fluid node. Or equation 2.35 for $\Delta\delta_x < 0.75\delta_x$ in order to improve numerical stability. [12]

$$\bar{\mathbf{u}}_{w2} = \frac{\left(2\mathbf{u}_b + (\Delta - 1)\,\mathbf{u}_{ff}\right)}{1 + \Delta} \tag{2.35}$$

Where $\mathbf{u}_{ff} = \mathbf{u}(\mathbf{r}_{ff})$, the velocity at the far fluid node with $\mathbf{r}_{ff} = \mathbf{r}_f + \mathbf{c}_i\delta_x$.

$$\bar{\boldsymbol{\rho}}_w = \boldsymbol{\rho}_w + \delta_x\mathbf{c}_i\nabla\rho = \rho_w + \epsilon\,(\mathrm{Ma}) \tag{2.36}$$

Finally, the non-equilibrium part is given by 2.37 for $\Delta \geq 0.75$.

$$f_i^{\mathrm{neq}}(\mathbf{r}_w, t + \delta_t) = f_i^{\mathrm{neq}}\left(\mathbf{r}_f, t\right) \tag{2.37}$$

Or equation 2.38 for $\Delta\delta_x < 0.75\delta_x$ in order to improve numerical stability. [12]

$$f_i^{\mathrm{neq}}(\mathbf{r}_w, t + \delta_t) = \Delta f_i^{\mathrm{neq}}\left(\mathbf{r}_f, t\right) + (1 - \Delta)\, f_i^{\mathrm{neq}}\left(\mathbf{r}_{ff}, t\right) \tag{2.38}$$

Finally, the post collision particle distribution is obtained as shown by equation 2.39.

$$f_i'(\mathbf{r}_w, t + \delta_t) = \bar{f}_i^{\mathrm{eq}}(\mathbf{r}_w, t + \delta_t) + (1 - \omega)\, f_i^{\mathrm{neq}}(\mathbf{r}_w, t + \delta_t) \tag{2.39}$$

All populations are now known, but the wall is still not moving. That is where the Inamuro iteration comes in.

Inamuro Iteration

First the streaming step from equation 2.19 needs to be corrected for an external body force as shown by equation 2.40.[14]

$$f_i(\mathbf{r} + \mathbf{c}_i\delta t, t + \delta t) = f_i'(\mathbf{r}, t + \delta t) + 3\delta_x\mathbf{w}_i\mathbf{c}_i\mathbf{g}(\mathbf{r}, t + \delta_t) \tag{2.40}$$

Where $\mathbf{g}(\mathbf{r}, t + \delta_t)$ is obtained through the Inamuro Iterations [14]. First an initialisation step is required in order to compute the initial value of the body at the boundary Lagrangian points as shown by equation 2.41. [14]

$$\mathbf{g}_0(\mathbf{R}_k, t + \delta_t) = \frac{\mathbf{U}_k - \mathbf{u}'(\mathbf{R}_k, t + \delta_t)}{\delta_x} \tag{2.41}$$

Where $\mathbf{u}'(\mathbf{R}_k, t + \delta_t)$ is obtained by interpolation as shown in equation 2.42. [14]

$$\mathbf{u}'(\mathbf{R}_k, t + \delta_t) = \sum_r \mathbf{u}'(\mathbf{r}, t + \delta_t)\, W(\mathbf{r} - \mathbf{R}_k)\, \delta_x^d \tag{2.42}$$

Where $\mathbf{u}'(\mathbf{r}, t + \delta_t)$ is known since the guo method obtained $f_i'(\mathbf{r}_w, t + \delta_t)$, W is a weighing function as proposed by Peskin [23] and d is the number of dimensions.

Now the iterations can be begin. Step 1: computation of the body force at Eulerian grid points at the l-th iteration with use of equation 2.43.[14]

$$\mathbf{g}_l(\mathbf{r}, t + \delta_t) = \sum_{k=1}^{N} \mathbf{g}_l(\mathbf{R}_k, t + \delta_t)\, W(\mathbf{r} - \mathbf{R}_k)\, \Delta V \tag{2.43}$$

Where ΔV is a small volume taken from the body surface. Step 2: correction of the velocity at Eulerian grid point as shown by equation 2.44. [14]

$$\mathbf{u}_l(\mathbf{r}, t + \delta_t) = \mathbf{u}'(\mathbf{r}, t + \delta_t) + \delta_x\mathbf{g}_l(\mathbf{r}, t + \delta_t) \tag{2.44}$$

Step 3: interpolation of the velocity at the Lagrangian boundary point as shown by equation 2.45.[14]

$$\mathbf{u}_l\left(\mathbf{R}_k, t+\delta_t\right) = \sum_r \mathbf{u}_l\left(\mathbf{r}, t+\delta_t\right) W\left(\mathbf{r}-\mathbf{R}_k\right) \delta_x^d \tag{2.45}$$

Finally, correction of the body force in step 4 which is also the last step as shown by equation 2.46.[14]

$$\mathbf{g}_{l+1}\left(\mathbf{R}_k, t+\delta_t\right) = \mathbf{g}_l\left(\mathbf{R}_k, t+\delta_t\right) + \frac{\mathbf{U}_k - \mathbf{u}'\left(\mathbf{R}_k, t+\delta_t\right)}{\delta_x} \tag{2.46}$$

From studies done by Inamuro it has been concluded that $l = 5$ is sufficient to ensure no-slip condition on the boundary points.

This concludes the theory explanation. In the next chapter the actual model and its code shall be discussed.

3

Model

3.1. Palabos

As stated earlier in the introduction, Palabos is open-source LBM implementation which is built on top of another project called OpenLB. Palabos is an collaboration project between industry (FlowKit) and academia (University of Geneva). In this section a more detailed picture will be given of what the Palabos Library has to offer.

3.1.1. Description

In Palabos the lattice is subdivided in multiple levels as shown by figure 3.1. This figure shows how the core of the library is built up. There are two ways of reaching the lowest level (Block3D) namely: clockwise or counterclockwise from the MultiBlockLattice3D which in LBM language is the lattice in 3 dimensions.

The clockwise direction shows the basis of the building blocks of the lattice that actually hold the data and processors. The MultiBlockLattice3D has a map of multiple BlockLattice3D. Each BlockLattice3D has a pointer to the dynamics which is the BGK collision operator as was shown by equation 2.24 in the theoretical chapter 2. Each BlockLattice3D also has a pointer to the raw data and grid coordinates. The BlockLattice3D is built on top of the class AtomicBlock3D by inheritance.
The AtomicBlock3D has the BlockStatistics which are the macroscopic variables and a StatSubscriber3D which controls the access to these statistics. Finally, it has an array of pointers to the internal processors. The AtomicBlock3D is built on top of Block3D which is the most basic level class.

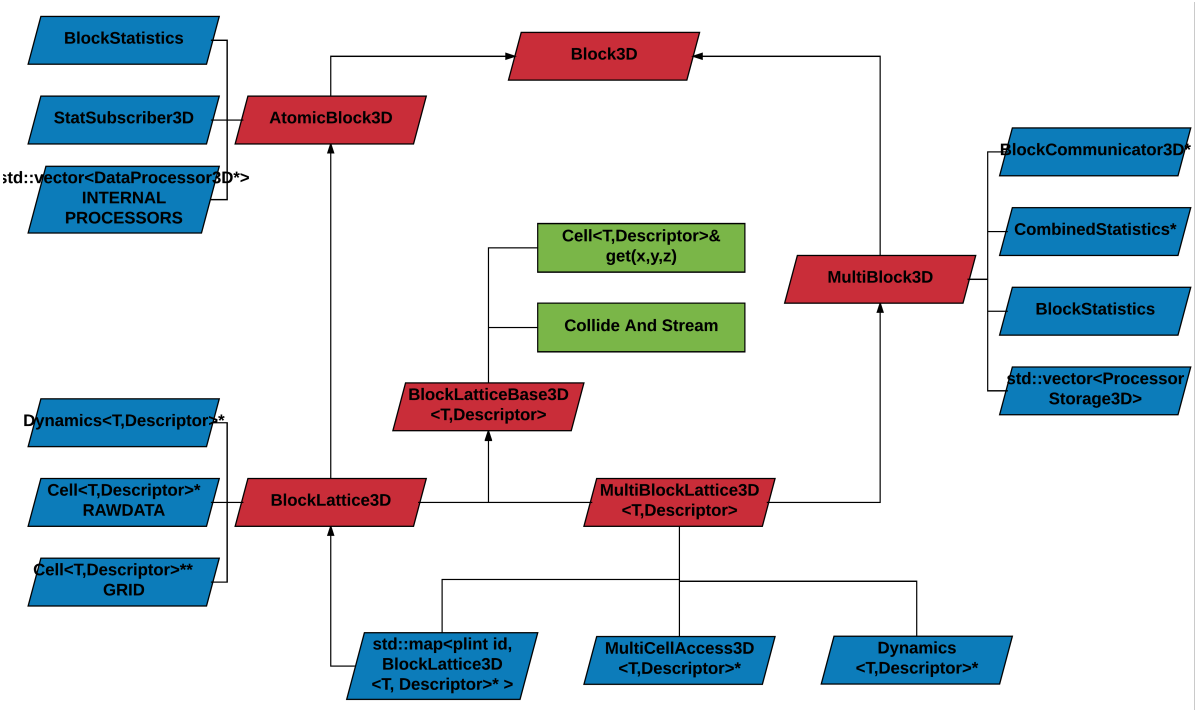


Figure 3.1: General Palabos Layout

The right hand side moving counterclockwise also starts at MultiBlockLattice3D which is built on top of MultiBlock3D. MultiBlock3D can be seen as the communication handler of the data on the left hand side. It has a pointer to the BlockCommunicator3D which does exactly that, a pointer to the CombinedStatistics which is created from the BlockStatistics. Finally, data storage is handled by the vector of ProcessorStorage3D.

In the middle of figure 3.1 the BlockLatticeBase3D is shown which initiates the collide and stream steps that where presented by equations 2.18 and 2.19. BlockLatticeBase3D also holds the method to access a certain cell in the lattice which is frequently called within the library: ”the main cell access method”.

Now that the core of Palabos is discussed, the following will describe some of the boundary conditions that are available.

3.1.2. Boundary Conditions

As stated in the theoretical chapter, the boundary condition chosen is the Guo extrapolation with an Inamuro Iteration to calculate the fluid force on the object so that the object can move. This boundary condition is chosen due to multiple reasons, but the main ones are that the Guo extrapolation also has a function to calculate fluid force and this boundary condition was also used in one of the few off lattice 3d examples. Since the documentation on off-lattice boundary conditions was absent, the knowledge about the boundary conditions had to be taken from these examples.

Other off-lattice boundary conditions that are available are the Filippova-Haenel boundary condition [10], Bouzidi boundary condition [5] and the Inamuro boundary condition [14].

3.1.3. Parallelization

As stated earlier the Lattice Boltzmann Method is extremely parallel. This feature is exploited in Palabos with an Message Passing Interface (MPI). It is essentially a software package that structures communication between network connected machines that run the same program in parallel. The lattice workload is divided over the machines running the simulation in a network through the split up as

described in the description of this section.

3.2. Code Dependencies

This section will elaborate on the software dependencies of the library and whether these are standard library dependencies or something that was introduced to improve the model.

3.2.1. CMake and GNU Make

The Palabos library comes with a scons software building script written in Python. But this program soon turned out to be very slow and is therefore unpractical to use in software development since testing requires numerous builds. Therefore a more versatile solution is proposed namely: the CMake and GNU Make combination. CMake builds the C++ files into the required system components whereas Make builds these components into an executable. The multithreaded cousin of GNU Make called Ninja was also considered but since this was not available this option was dropped and the GNU Make multithread -j option was used instead.

3.2.2. TinyXML

In order to read parameter xml files in C++, Palabos includes the TinyXML header directly into it's library. TinyXML is a fast and easy to use program therefore no alternatives need to be considered.

3.2.3. OpenMPI

OpenMPI stands for Open source Message Passing Interface and is the software that is used to make used of Palabos' parallelisation. It is not directly included into the library but the library code is written to be built with it through the use of macros when the -DPLB_MPI_PARALLEL definition is included in the CMake options.
MPI works by structuring messages over a network distributed set of machines. This structuring means consistency but also synchronisation when required. Its speed is limited by the amount of machines and the maximum level of parallelisation in the code. In general it is much faster than simple multi-threading but it is also very difficult to debug.

This concludes the code dependencies. The list is rather small but is mostly due to the nature of C++. C++ lets one use all the available system components that would be out of reach in other languages.

3.3. Challenges

When creating the model of this thesis some design and software challenges arose. This section will elaborate on those, thereby giving an idea of the work flow of the model constructing process.

3.3.1. Software Challenges

Palabos uses an scons distribution software to build the code. Since scons is written in Python and extremely slow this had to be changed to GNU make.
In order to use the C++11 coding language the proper C-library needs to be installed, which was not a problem on a local machine but proved somewhat difficult to install on the cluster or hpc11.tudelft.net where hpc stands for High Performance Computing (HPC). This difficulty was due the administrative and technical problems regarding the C-library update. Building the library manually worked but the MPI compiler could not find it. Hence the model was built and tested locally for the majority of the time until a TU wide system upgrade during the summer provided the correct library, namely: the version of 2011. Open-MPI was already installed and with the systems upgrade was also upgraded to a newer version.
In order to perform memory debugging normally, an open-source program can be used, but since the model is highly parallel different debugging software is required. There are two main solutions namely: TotalView and Allinea Forge which are both extremely expensive and therefore not owned by the university. In order to overcome this problem a workaround was used by creating a handler for the SIGSEV signal and other signals that can be thrown, the handler would freeze the MPI processes and one can then attach a serial debugger to the process which threw this signal. One downside was that on the HPC server the process folder can only be used with administrator privileges to which access was not

granted. A third option was also considered namely the so called Code Sanitizers. Clang which is the Mac OSX C++ library includes a Memory and Address Sanitizer. The GNU C++ library on the other hand only includes an Address Sanitizer. Sanitizers work by attaching the debug program in the building phase thereby eliminating the need for an additional debug run. Next to that they are very fast and extremely easy to use. Unfortunately the GNU sanitizers were only included in the latest C++ library and the network administrator did not want to install the latest C++ library hence this option was finally dropped. Thus the debug option used was the standard serial debugger Valgrind on the HPC which is exceptionally slow since it is serial and locally the CLang Sanitizers were chosen. Hence the model needed to be built and tested locally which proved time consuming.

Finally, due to the nature of an open-source project namely multiple contributors over a long length of time, the library is written in multiple c and c++ languages, making it hard to read and back track problems. This was especially a problem with the off-lattice boundary conditions since there was no available documentation hence these needed to be learned from the code itself. Next to the use of multiple languages there is no or little code formatting consistency throughout the library.

3.3.2. Design Challenges

As stated earlier there is no available documentation on the off-lattice boundary conditions in Palabos. This made it hard to evaluate which boundary condition would prove best for the problem at hand. By reverse engineering the examples and the library code some knowledge could be obtained.

The most difficult part in creating the model was to merge the various techniques that are available in the library, namely: a moving object in a flow, an object within an enclosed container and finally execute the model in three dimensions. The examples shown on the website of Palabos show an example of a moving object in a flow. This example however, is not included in the library. There is an example of a moving wall but this moving wall is a simple rectangular shaped on-lattice boundary condition. On the other hand there exists an example of an off-lattice boundary condition namely the aneurysm example but this example does not include a moving object.
Combining the two proved difficult since it is unknown if those two algorithms are meant to work together or not.

3.4. Code Standardisation

In order to create some consistency for use in the future the following code standardisation is proposed.

- Always use two header files namely the .h for definitions and .hh for implementation
- Never include more then one class in one header file pair
- Give the files the name of the class
- Always include a macro header guard at the beginning of the header file
- If a function in one class does the same as in another class give it the same name
- Standardise exception handling by writing one exception handler
- Include the exception handler in every function
- Include debug information in every function
- Make debug information visible through use of a macro
- Consider static allocation of as much of your code to the stack
- Keep the main.cpp file short and clear to read

These code standardisation rules were learnt during this thesis project. Hence they might not be applied everywhere in the code where they should have been. If there would have been more time available this would have been changed in the code together with the removal of abundant code.

3.5. Code Implementation

This section will elaborate on the code. In order to keep this section readable, all code is placed in the appendix and will be discussed briefly. To understand how the main file works other classes need to be explained first.

3.5.1. CMake

CMake is the only piece of code which is not a class. In order to be able to deploy the code on multiple platforms namely hpc11.tudelft.net and locally on a MacBook, a CMake script must be written in a way that it works on all machines. Code dependencies need to be searched on the system and since those systems differ, full paths cannot be used.

Appendix A.1 shows this CMake script. Lines starting with a hashtag are commented out. The script consists of roughly four parts. The first part searches the system for the project dependencies. The second part turns options on or off either manually or automatically. The third part searches for the files of the library that needs to be built, as well as other CMake files. There is one other CMake file which only constructs the palabos library and is also included in this appendix section. Lastly the fourth part links and creates the executable.

3.5.2. Helper

Appendix A.4 contains the helper class. This class is split up in compliance with the earlier proposed code standardisation rules. The helper.h file contains the class function and attribute definitions, with the helper.hh contains their implementation.

The purpose of this class is to control the state of all the other classes. It creates and destroys, them eliminating the need to do so per class when required. The class has it's own constructor and destructor which creates and destroys all the other classes respectively. This class also contains one function initialise which initialises the other classes where required with a parameter filename. Simulation parameters are placed in a separate settings file for ease of use.

The helper class has all the other class instances as its attributes. This class also has an object counter which is used to simulate a static class since a static class is not allowed in C++. Lastly the class has a boolean to order which MPI process is sending messages to the console. By doing this only the master process communicates with the user.

3.5.3. Constants

The constants class, as the name suggests, holds all the constants required for the simulation. This class is located in appendix A.5.

The class is initialised with a parameter xml filename as explained before. In this initialisation deduced constants are also determined and the compressibility errors are checked for all simulation settings. This class also has a create minimal settings function but which was intended to create the optimal settings for a simulation. This however, is still under construction.

3.5.4. Obstacle

The obstacle class in appendix A.6 was created to hold all data with respect to the obstacle in the flow which is in this case was a sphere. This class also has an initialise function but now reads an STereoLithographic (STL) file which contains the object 3D geometry.

The obstacle class has 6 functions not including the constructor, destructor and the initialiser function.

Get Centre

This function calculates the centre of the obstacle from its vertices either with a triangle set as input, or without.

Get Domain
This function computes the smallest bounding box from it's vertices again either with a triangle set as input or without. This function is especially useful when checking if the obstacle is out of bounds.

Get Volume
Calculates the volume by calculating the tetrahedron volume. The tetrahedron is taken with the base as the surface triangle and the top as the centroid. This volume is then used to determine the mass of the object.

Move to Start
This function is used to move the obstacle to it's starting position in the lattice thereby not placing it outside the container with help from the get domain function.

Update Immersed Wall
This function is necessary in order to update the new locations of the vertices in the inamuro iteration. This needs to be done each step in order to move the obstacle through the flow.

Move
Perhaps the most important part of this class is the move function, which calls the inamuro iteration and determines the force and torque. After that, it passes this information to the velocity function which calculates the new speed of the object. It must be noted that the reduceImmersedForce and reduceImmersedTorque are the two functions which don't return values after a code update. The source for this problem is currently not known.

The attributes of this class are mostly pointers which are explained in the variables class.

3.5.5. Wall
The wall class from appendix A.7 is actually improperly named, since it is not a wall but the fluid container. This class again has the same object counter, constructor and destructor as all the other classes and those properties. For clarity, future explanations will not include those anymore since they are the same everywhere. This class is also initialised with a STL file and has two functions.

Get Centre
This function is used to determine the obstacle starting position at the top centre of the container.

Get Domain
The get domain function is used to monitor if the obstacle is out of bounds.

3.5.6. Variables
The variables class is perhaps the most important class. It creates the palabos simulation environment through 18 functions as shown by appendix A.8.

Update
This function updates all the necessary parameters in case either the grid level (level of grid refinement) or Reynolds number are changed.

Check Convergence
This function checks wether the simulation has converged an can be stopped. In test mode this function is not used since the simulation then stops after a certain amount of iterations.

Get Rho
This function was solely intended for verification purposes to check if the flow density from the simulation is similar to that found in literature. [2]

Create Mesh
This function turns the triangle set that was read from the STL file into a mesh by creating a palabos class named DEFScaledMesh which is the triangle set extended in outward and inward directions by a given number of layers.

Create TB
TB stands for Triangle Boundary as such this function creates the triangle boundary by taking the DEFScaledMesh class as it's argument. Thus a new palabos class is created named TriangleBoundary3D and is just a necessary intermediate step.

Create Voxels
This function takes the previous TriangleBoundary3D instance as it's argument and voxelises it. The so called "voxelisation process" determines which side faces the fluid and which side faces the solid for each of the planes bounded by the vertices. This information is then stored in so called Voxel Flags and this function then sets the nodes containing the solid to constant. The resulting class of the voxelisation process is the palabos class VoxelizedDomain3D.

Create BP
BP stands for Boundary Profile and takes the now voxelised TriangleBoundary3D as its argument. Now that the profile off the boundary is defined, a no slip velocity profile is added by this function. Resulting in a new palabos class called BoundaryProfile3D.

Create FS
This function takes both the VoxelizedDomain3D and BoundaryProfile3D as it's arguments. Create FS stands for create Flow Shape. Since the surface in this thesis consists of triangles, a Palabos class TriangleFlowShape3D is created by this function.

Create Model
The create model function creates the Palabos class GuoOffLatticeModel3D which is actually the boundary condition explained in the theory. This function takes two arguments: the TriangleFlowShape3D and a integer flowType which indicates if this boundary is an outside flow or inside flow boundary.

Create Lattice
This function takes two voxelised domains as arguments namely: that of the wall class and that of the obstacle class. It then copies the obstacle information into the wall information and uses the combined data to create the lattice or in Palabos: the MultiBlockLattice3D. The LBM parameters are then again calculated and written to a log file in order to make sure everything was created correctly. Next, the J and RhoBar data fields are created to hold the macroscopic variables which are then merged into the so called "RhoBarJarg". Two processing functionals are created: one for the collide and stream, and one to determine the value for the RhoBarJarg. Processing functionals are extra defined LBM functions that are executed after the collision and streaming step. Next, the vertices are obtained from the mesh in order to create yet another functional namely: the Inamuro iteration which is called by the Palabos function: InstantiateImmersedWallData3D. This is followed by a number of Inamuro processing functionals.

Create BC
BC stands for Boundary Condition and is the palabos top level class for the boundary condition. It is created from the GuoOffLatticeModel3D and the voxelizedDomain3D. If another boundary condition would be chosen, another model should be inserted here. The result of this function is the OffLatticeBoundaryCondition3D with the RhoBarJarg inserted.

Initialise Lattice
The lattice is initialised at thermal equilibrium and all processing functionals are executed ones. The lattice is now ready for the simulation.

Make Parallel
The Make Parallel function was intended to re-parallelise the entire lattice after formation but is in it's current form not used.

Set Lattice
The set lattice function is a helper function which executes previous functions in the right order for the wall and obstacle.

Save
This function calculates and stores the flow velocity, vorticity, and density, however since these are large datasets and already a lot of memory is used by the program, this function is turned off and the data is written to the hard drive by a function in the output class.

Update Lattice
This function is the handler for the save function.

3.5.7. Output

The output class handles indeed the output of the simulation as shown by appendix A.9. It has 10 functions to handle the data written to the hard drive and the debug messages written to the console.

Elapsed Time
Writes the elapsed time to console.

Time Loop
The Time Loop function is a function for use in the test environment. It checks whether the elapsed time exceeds the maximum test time set by the constants class and if so, stops the simulation. This function can be used when the iteration limitation in the test mode takes to long to stop the simulation.

Write GIF
The Write GIF function writes the velocity of a slice of the 3D simulation to a GIF file. Due to complications with GIF image writer, this function is currently not used.

Write Density
The Write Density function computes the density and writes it to a Visualisation ToolKit (VTK) on the hard drive.

Write Velocity
This function computes the velocity and writes it to a VTK file on the hard drive.

Write Vorticity
The Write Vorticity function computes the vorticity and writes it to a VTK file on the hard drive.

Write Images
This function is a combination of the above functions and writes the density, velocity, and vorticity simultaneously.

Start Message
This function writes a start message at the beginning of the simulation to the console before creating all the classes.

Sim Message
The Sim Message function writes a message to the console after all classes are initialised and actual simulation can start.

Stop Message
The last function of this class is the Stop Message function and writes a message to the console when the simulation ends.

3.5.8. Surface Velocity

The Surface Velocity class as shown by appendix A.10 holds and computes the velocity data for the immersed object and is called by the Inamuro Iteration. One of the other options were the model currently fails could also lay within this class. It has one operator and 9 functions.

Operator ()
This operator is called by the Inamuro iteration and returns the velocity of a vertex with int id.

Get Domain
This function is the same as the one in the obstacle class but since no instances of this class are available here it is implemented again.

Get CG
This function calculates the centre of gravity of the object with the assumption that the centre of gravity coincides with the centre of mass and use of equation 2.10.

Get Arm
Get Arm is a function that calculates the line segment from one point to another.

Get Moment of Inertia
This function calculates the moment of inertia with use of equation 2.12 for every inward facing tetrahedron and its top coincides with the centre of gravity or centroid. Summing all tetrahedrons up results in the moment of inertia matrix of the object.

Get Alpha
This function computes the angular acceleration from equation 2.9 with use of torque on the object and moment of inertia of the object.

Get Rotation
The Get Rotation function rotates the object about an angle $\partial\theta$ about its centroid.

Get Total Velocity
This function adds the angular velocity and linear velocity for a given vertex.

Out of Bounds
The Out of Bounds function checks wether the object is out of bounds. If so the simulation is stopped.

Update
This is the handler to all above functions. It takes the fluid force, fluid torque, the Incompressible Flow Parameters, and the TriangleBoundary3D as it's arguments. It first makes a list of all vertices by copying them from the mesh. Next, the gravity force on the object is calculated from it's mass. This gravity force is then added to the fluid force and the linear acceleration can now be calculated. From the torque the rotational acceleration is calculated. Now, velocities are obtained by adding the new velocity increment to the previous one. Finally, for each vertex the total velocity is obtained and the maximum of it's values is saved for debugging purposes.

The mesh is then rotated and translated through the fluid and all variables are converted to physical units and written to the console.

3.5.9. Normal
This class shown in appendix A.11 is similar to the velocity class although much simpler and is required for the inamuro iteration. It has only one operator and one function.

Operator ()
This operator returns the normal for a vertex with int id and is also called by the inamuro iteration but this time to calculate the torque.

Update
This function calculates the new normals for a given TriangleBoundary3D.

3.5.10. MPI Data Manager
This class is an extension to the MPI manager already in the palabos library. It uses more top level classes instead of basic operations, for instance: instead of sending a scalar it sends a 3D field of scalars. Its code can be seen in appendix A.12. The MPI Data Manager class has 6 functions and a friend instance called mpiData for easy access.

Send Scalar Field
The Send Scalar Field function sends a field of scalars using the Message Passing Interface with synchronisation.

Receive Scalar Field
This function receives a field of scalars using the Message Passing Interface with synchronisation.

Send Triangle Set
Send Triangle Set sends a set of triangles using the Message Passing Interface with synchronisation. This function and the next were necessary since machines running on another server might not be able to access the hard drive containing the STL files.

Receive Triangle Set
This function receives a set of triangles using the Message Passing Interface with synchronisation.

Send Receive Domains
This function communicates between all machines on which domain they are currently operating and if they overlap or form gaps an exception is thrown.

Split Domains
This function calculates the domain in which each machine should work.

3.5.11. Exceptions
These are multiple classes extending the exception class. Doing so allows the throwing of newly created exceptions. The following exceptions were created.

- Compressibility Error too Large
- Local Compressibility Error
- Super Sonic Flow Error
- Margin smaller than border width Error
- Resolution Error

The last two exceptions were created to avoid entering buggy parameters into the parameters XML file.

3.5.12. Debug
This is not a class but a group of debug functions created to improve the debugging information on machines without a debugger. It must be stated that this class should be rewritten in order to comply with the code standardisation rules.

Print Trace
This function prints the stack trace of process which threw an exception.

Exception Handler
The Exception Handler function or “exHandler” is the function that standardises how exceptions are thrown.

Memory Usage
This function is also called “memUsage” gives a very complete insight in the memory usage of the simulation.

Signal Handler
The Signal Handler or “sigHandler” handles signals that are usually thrown by the system in case something goes wrong but is not caught by the try catch and exception.

Install Signal Handler
Install Signal Handler installs the Signal Handler with the signals for that specific operating system.

Wait GDB
GNU DeBugger (GDB) is a serial memory debugger and as explained earlier, a parallel one was not available. This function writes the Proces IDentification (PID) number to the console so that GDB can be attached to that PID.

Address to String
This function is a memory safe implementation to write a memory address to the console.

Safe String
Safe String is a memory safe implementation for a string.

Array String
This function is a memory safe implementation to write an array to a string.

Box String
Finally Box String is a memory safe implementation to write a 3D box (Domain) to a string.

3.5.13. Geo
This class shown in appendix A.15 holds basic geometric operations for points, triangles and tetrahedrons.

3.5.14. Main
Since all classes and functions have been explained, a stepwise representation of the simulation process once, as presented by main in appendix A.2 will ensure.

1. Install Signal Handler
2. Initiate Palabos
3. Initialise the Helper class
4. Get pointers to all the objects
5. Write messages to console
6. Start looping over Reynolds number
7. Start looping over grid level
8. Execute the data processors
9. Move the obstacle
10. Update the Lattice
11. Write Debug Info
12. Write Data to Hard Drive

This completes the model code. To know whether the model is correct, it must be checked through verification.

4

Verification and Validation

As stated in the preceding chapters, due to the nature of the project, many technical as well as administrative restrictions and limitations constrained the outcome of this research. These, as well as time constraints, resulted in that the simulation was not able to produce conclusive results that in turn could be verified. Due to these reasons, this chapter shall describe the necessary calculations that would have been used to verify and validate the results, would the limitations have been overcome. Although limited in this thesis, these calculations should be seen as a presentation of how future research may go about in both verification and validation regarding a project of this nature.

4.1. Verification

Let the fuel salt for the verification be fuel salt 1 from table 1.1 with the thermodynamic properties given by table 1.2. A verification temperature of 1000 Kelvin is proposed since all properties from table 1.2 are valid for this temperature. The dynamic viscosity can then be calculated with equation 4.1. [2]

$$\mu_f = \exp\left(0.178 + \frac{403.5}{T - 804.2}\right) \approx 9.3818[mPa \cdot s] \tag{4.1}$$

In the same manner, the fluid density can be calculated as shown by equation 4.2 also obtained from 1.2. [2]

$$\rho_f = 5155 - 0.8331 * T = 5155 - 833.1 \approx 4321.9[kg/m^3] \tag{4.2}$$

In order to verify the model the terminal velocity is calculated. This shall be done for the three flow region described in section 2.2.1.

4.1.1. Stokes Flow

Stokes flow is valid in the region Re< 1, in order to remain well within this region, let's set the Reynolds number to $Re = 0.5$ and let the characteristic velocity be the stokes terminal velocity given by equation 4.3. [15]

$$v_s = \frac{2}{9}\frac{(\rho_s - \rho_f)}{\mu_f} g R^2 \tag{4.3}$$

Let the characteristic length scale be the diameter of the sphere. Then by rewriting equation 2.27 to equation 4.4 will yield the characteristic velocity which will be set equal to the terminal velocity.

$$v_s = \frac{\mathrm{Re}\mu_f}{\rho_f 2R} = \frac{0.5 \cdot 9.3818 \cdot 10^{-3}}{4321.9 \cdot 0.02} \approx 5.4269 \cdot 10^{-5}[m/s] \tag{4.4}$$

The required density of the sphere to reach this velocity can be obtained by rewriting equation 4.3 to equation 4.5.

$$\rho_s - \rho_f = \frac{9 v_s \mu_f}{2 g R^2} = \frac{9 \cdot 5.4269 \cdot 9.3818 \cdot 10^{-8}}{2 \cdot 9.81 \cdot 0.01^2} \approx 0.002[kg/m^3] \tag{4.5}$$

Now, by setting the above obtained parameters as the simulation parameters and increasing the container size in the simulation parameters so that an infinite fluid is approximated, the terminal velocity should be approached in the simulation and the model would thereby be verified for the lower Reynolds numbers. If this verification would fail an alternative approach could be used by choosing an arbitrary density for the sphere which is significantly larger.

4.1.2. Schiller-Naumann

Schiller-Naumann's empirical formula as shown by equation 2.16 is valid in the region $1 <$Re< 1000. Hence a value of $Re = 500$ is chosen to remain well within this region. Balancing forces, substituting Schiller-Naumann's in this equilibrium and rewriting for the terminal velocity the following equation is obtained (equation 4.6).

$$v_s = \sqrt[1.687]{\frac{2/9R^2 g\left(\rho_s - \rho_f\right)}{\mu_f + 0.15\left(2\rho_f R\right)^{0.687}}} \tag{4.6}$$

The required terminal velocity is calculated in the same manner as done by equation 4.4 and is shown by equation 4.7.

$$v_s = \frac{\text{Re}\mu_f}{\rho_f 2R} = \frac{500 \cdot 9.3818 \cdot 10^{-3}}{4321.9 \cdot 0.02} \approx 5.4269 \cdot 10^{-2}[m/s] \tag{4.7}$$

The required density of the sphere to reach this velocity can be obtained by rewriting equation 4.6 to equation 4.8.

$$\rho_s - \rho_f = \frac{9v_s^{1.687}\mu_f + 1.35v_s^{1.687}\left(2\rho_f R\right)^{0.687}}{2gR^2} \approx 180.30[kg/m^3] \tag{4.8}$$

Now, by setting the above obtained parameters as the simulation parameters and increasing the container size in the simulation parameters so that an infinite fluid is approximated, the terminal velocity should be approached in the simulation and the model would thereby be verified for the intermediate Reynolds numbers.

4.1.3. Newtonian

Newtonian drag was presented in the theory by equation 2.17 and is valid for Re> 1000. An arbitrary Reynolds number of Re$= 1500$ is chosen for the verification of this Reynolds number range. Then by again balancing forces of the settling sphere, inserting the Newtonian drag and rewriting for the terminal velocity yields equation 4.9.

$$v_s = \sqrt{\frac{16Rg\left(\rho_s - \rho_f\right)}{3\rho_f C_D}} \tag{4.9}$$

The required terminal velocity is calculated in the same manner as done by equation 4.4 and is shown by equation 4.10.

$$v_s = \frac{\text{Re}\mu_f}{\rho_f 2R} = \frac{1500 \cdot 9.3818 \cdot 10^{-3}}{4321.9 \cdot 0.02} \approx 0.1628069[m/s] \tag{4.10}$$

The required density of the sphere to reach this velocity can be obtained by rewriting equation 4.9 to equation 4.11 and using $C_D \approx 0.44$ for a sphere as stated in the theory.

$$\rho_s - \rho_f = \frac{3v_s^2 \rho_f C_D}{16Rg} \approx 96.341[kg/m^3] \tag{4.11}$$

Finally, by setting the above obtained parameters as the simulation parameters and increasing the container size in the simulation parameters so that an infinite fluid is approximated, the terminal velocity should be approached in the simulation and the model would thereby be verified for the higher Reynolds numbers. If this verification would fail an alternative approach could be used by choosing an arbitrary density for the sphere which is significantly larger.

4.2. Validation

As stated earlier since there were no results from the simulation, validation is not possible. The proposed validation would be to compare the simulation to the measurements done by ten Cate et al. [32].

Again, one must stress that the limitations in this research, although resulting in the absence of simulation data for this project in particular, do add to the academic discourse by presenting methodologies that may be used in future research. A discussion of what contributed to the absence of data will be detailed in the chapter on results, Chapter 6.

5

Simulation

This chapter will describe the simulation that the model was designed to execute. As stated earlier the sphere and container were created with a Mesh program generating an STL file as shown by figure 5.1. The Mesh programs used were GMesh and MeshLab.

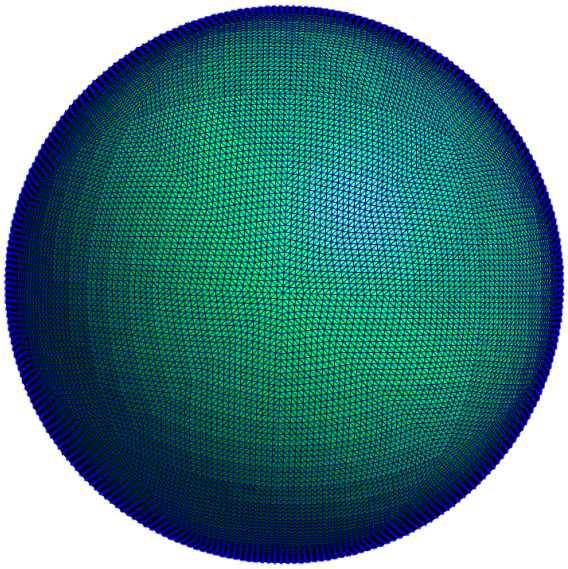


Figure 5.1: The STL of the sphere

This sphere is loaded in the model as is the container as shown by figure 5.2. The sphere is then moved to its starting position and the boundary conditions are applied as described in section 3.5.6. The Reynolds number is then incremented, and for each Reynolds number a simulation is run until convergence has been reached.

Figure 5.2: The STL of the container

Since the model is still in its testing phase some final changes have not been made yet. The Reynolds number should be a double and a smaller increment should be used (e.g. $\partial \mathrm{Re} = 0.01$). The grid refinement has also not yet been implemented.

The model did not run correctly since no fluid force and torque could be calculated. Thus the question that remains to be answered is whether anything went wrong: an identification of inaccuracies. Chapter 6 will elaborate further on this.

6

Result

As stated before the model was unable to produce the required results. This chapter will investigate the source of the problem.

6.1. Log

In appendix B, a section of the log of one of the latest runs is shown. When looking at this log it becomes clear that the initialisation of the model goes without complications. The model also runs correctly and the sphere moves through the fluid. The fluid force and torque acting on the sphere on the other hand remain zero throughout the simulation. It is feasible that they remain zero for the first couple of iterations, but after 2 iterations there should be a change. The only reason the obstacle moves is because of the gravity force imposed on the sphere. What could have caused the fluid force and torque to remain zero?

6.2. Possible Causes

The fluid force and torque are calculated through the inamuro iterations. The following steps take place in setting up the data and these iterations themselves.

6.2.1. Initialization

Functions in Palabos can be executed directly or can be executed through processing functional. The processing functional in this case is called **integrateProcessingFunctional** and takes a pointer to a class called **InstantiateImmersedWall** together with the following input.

- Box3D domain
- MultiContainerBlock3D container

Here Box3D domain is bounding box of the object to do the calculations on and the MultiContainerBlock3D container is the handle to the actual MultiBlocks and thereby the cells to use in the calculations. The InstantiateImmersedWall class has the following input.

- Vertices
- Areas
- Unit Normals

Here these vertices, areas and unit normals are those of the surface of the sphere.

The only possible cause in the initialisation is the container, since this holds the actual cell data. If no macroscopic variables are recovered from the fluid surrounding the surface of the sphere then no force and torque can be calculated.

6.2.2. Iteration

The inamuro iteration is again executed through a processing functional which takes a pointer to the class **IndexedInamuroIteration** together with the following input.

- Box3D domain
- MultiScalarField3D rhoBar
- MultiTensorField3D J
- MultiContainerBlock3D container

Where the domain and container have already been discussed, rhoBar is used to manage the accuracy in the model and J is the first order velocity moment. The IndexedInamuroIteration class has the following input.

- SurfaceVelocity
- τ
- bool Incompressible = true

Where SurfaceVelocity is the velocity class created for this thesis as shown by section 3.5.8, τ is the relaxation time given by equation 2.26 and finally this thesis uses an incompressible model.

The same possibility exists as with the initialisation namely, that the macroscopic variables could not be obtained from the fluid represented by rhoBar, J, and the container. On the other hand, here the SurfaceVelocity could be at fault, since it is possible that the SurfaceVelocity does not provide the correct vertex velocity when it's operator() is called by the inamuro iteration. This seems the most likely error since in the log from appendix B there is a velocity of the object but the maximum vertex velocity remains zero. Thus there is a probability that a bug is the aforementioned section of the code, that adds the rational velocity to the linear velocity.

6.2.3. Force and Torque Calculation

Before the force is calculated the function resetForceStatistics is called, which has the previously mentioned container as it's argument. This function was advised by comments in the source code to be called in case the object did not move. Afterwards the function reduceImmersedForce is called which takes the following arguments.

- MultiContainerBlock3D container
- int VoxelFlag::outside

The container has already been discussed but the voxelFlag indicating whether a node lies in the fluid or in the solid is optional. Since the voxelFlag could well be wrongly defined in the voxelisation process, this would be a decent error in the problem as well. The torque is calculated in a similar manner and therefore skipped to avoid ambiguity.

6.3. Other Result

The additional result of this thesis is a model which although being a work in progress and thus inconclusive, still presents an alternative approach to a moving wall model in the LBM. Chapter 7 will move on to discuss the ways in which this thesis adds to academic discourse, and the benefits that further research into this subject provides.

7

Recommendation

Regarding further research on the simulation of fuel salt flow with the use of the Lattice Boltzmann Method, the following recommendations can be made.

7.1. C++

The versatility of C++ combined with it's speed makes it a good choice as the programming language for a LBM library, because templates, classes, unique pointers and other implementations provided by C++ would keep the library simple and improve readability if implemented correctly. In order to achieve this code standardisation rules should be implemented throughout the library and documentation should cover the entire library. Since C++ is still being developed today it would be recommend to keep the library updated with the latest version of C++ in order to benefit the most from new implementations and improvements.

7.2. Parallelisation

The cluster used in this thesis (hpc11.tudelft.net) has 48 processing threads per machine. Running OpenMPI on a single machine would be faster then doing so with multiple machines since the network connection would be the rate limiting step. If in future research larger problems needed to be modelled, a choice must be made. If speed is not a requirement then the same cluster could be used with more then one machine. If this proves to be too slow another option might be better. One of the options could be to search for a machine with more processing threads. Another option might be to use General Purpose computing on Graphical Processing Units (GPGPU) which is currently being researched by another section at the Reactor Institute Delft (RID). A detailed benchmark test would be needed to verify that the new hardware solution would prove superior, since LBM might prove to be unsuitable to benefit from the advantages GPGPU brings. Also it should be taken into account that parts of the library would need to be rewritten to work with GPGPU.

7.3. Software Library

Since the Palabos library does not have much consistency, nor a suitable amount of documentation, it would not be recommended to continue using the Palabos library in it's current form. A solution could be to set up a list of code standardisation rules and applying those to the library, together with simplification were possible and renaming were preferred. New documentation should then be created in order to provide future researchers with all the information they require regarding the LBM library.
Another solution could be to try the OpenLB library since Palabos is already built on top of that. This might provide a more structured and efficient library, but this together with the completeness of it's documentation should be studied further. Finally a comparison study could be done in greater detail to all available libraries and their documentation. In this study a benchmark test could be done for technical comparison of the libraries. A final choice can then be made on the result of this study taking into account that creating a new library is also an option. The end result would be a simulation platform

that the University can use free of charge with all documentation and rules available. It is strongly recommended to do this in C++ for the reasons stated earlier.

8

Conclusion

It is hard to give an elaborate conclusion to this thesis due to the fact that no results were generated. Hence conclusions towards the speed and accuracy of the library and model cannot be made.

It can be concluded though that the Palabos open-source library is difficult to use. First of all due to the fact that it is open-source: it is evident when reviewing the code that multiple programmers have worked on this project each coding in their own style and sometimes using different versions of C++, resulting in a hard to read incoherent library. Secondly due to the fact that the documentation is incomplete: although basic documentation is provided, documentation is lacking in the off-lattice part and on parallelisation. Thirdly due to the fact that very little debug information is implemented in the classes and functions making it hard to track bugs, the same holds for exception handling.

It can also be concluded that another parallelisation approach or hardware setup might be necessary in the future. Although still in debug mode, the current model had a runtime of about three days. Switching to a single machine reduced this to just one day but this will limit the amount of memory that can be used. Considering that the debug mode was limited to 20 collide and stream iterations one day of runtime is unacceptable. It must be stated though that other boundary conditions combined with a different force calculation iteration might have better speed and accuracy, but this needs to be researched further.

To conclude, the Palabos library might prove a good solution in the future if modified and documented. More research on different boundary conditions and their performance is required. More research is also required on the performance of the Palabos library compared to other libraries. Finally a performance analysis for hardware and parallelisation methods would be required to make a decision on improvements in that department.

A

Code Overview

CMakeLists.txt 07/12/2016, 13:32

A.1. CMake

```
#######################################
# CMAKE PALABOS MAIN
#######################################

# Force CMake version 3.1 or above
cmake_minimum_required (VERSION 2.8)

#set(CMAKE_C_COMPILER "/Library/Developer/CommandLineTools/usr/bin/clang")
#set(CMAKE_CXX_COMPILER "/Library/Developer/CommandLineTools/usr/bin/clang+
    +")

# use, i.e. don't skip the full RPATH for the build tree
#SET(CMAKE_SKIP_BUILD_RPATH  FALSE)

# when building, don't use the install RPATH already
# (but later on when installing)
#SET(CMAKE_BUILD_WITH_INSTALL_RPATH FALSE)

#SET(CMAKE_INSTALL_RPATH "${CMAKE_INSTALL_PREFIX}/lib")

# add the automatically determined parts of the RPATH
# which point to directories outside the build tree to the install RPATH
#SET(CMAKE_INSTALL_RPATH_USE_LINK_PATH TRUE)

# the RPATH to be used when installing, but only if it's not a system
    directory
#LIST(FIND CMAKE_PLATFORM_IMPLICIT_LINK_DIRECTORIES "$
    {CMAKE_INSTALL_PREFIX}/lib" isSystemDir)
#IF("${isSystemDir}" STREQUAL "-1")
#   SET(CMAKE_INSTALL_RPATH "${CMAKE_INSTALL_PREFIX}/lib")
#ENDIF("${isSystemDir}" STREQUAL "-1")


# Project name:
project(PALABOS)

# Palabos ROOT
set(PALABOS_ROOT ${CMAKE_CURRENT_SOURCE_DIR})

# Set Location of CMake scripts
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${CMAKE_CURRENT_SOURCE_DIR}/src/
    externalLibraries/")
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${CMAKE_CURRENT_SOURCE_DIR}/
    CMake/")

# Search for Sanitizers
find_package(Sanitizers)

#Add tinyxml from externalLibraries
include_directories("${CMAKE_CURRENT_SOURCE_DIR}/src/externalLibraries/
    tinyxml")
file(GLOB CXX_FILES "${CMAKE_CURRENT_SOURCE_DIR}/src/externalLibraries/
    tinyxml/*.cpp")
file(GLOB C_FILES "${CMAKE_CURRENT_SOURCE_DIR}/src/externalLibraries/
    tinyxml/*.c")
file(GLOB HH_FILES "${CMAKE_CURRENT_SOURCE_DIR}/src/externalLibraries/
    tinyxml/*.hh")
file(GLOB H_FILES "${CMAKE_CURRENT_SOURCE_DIR}/src/externalLibraries/
    tinyxml/*.h")
set(TINYXML_SOURCE ${CXX_FILES} ${C_FILES} ${H_FILES} ${HH_FILES})
```

```
add_library(tinyxml STATIC ${TINYXML_SOURCE})

#============================================================================
    ========================================
#
#       OPTIONS
#
#============================================================================
    ========================================

#Santizer Options
#option(SANITIZE_ADDRESS On)

SET(CLANG_LIB "/Library/Developer/CommandLineTools/usr/lib/")
SET(CLANG_INCLUDE "/Library/Developer/CommandLineTools/usr/include/")

#SET(GNU_LIB "/usr/local/Cellar/gcc/5.3.0/lib/")
#SET(GNU_INCLUDE "/usr/local/Cellar/gcc/5.3.0/include/")

IF(CMAKE_COMPILER_IS_GNUCXX)
    SET(COMPILER_FLAGS "-std=gnu++11")
ELSE(CMAKE_COMPILER_IS_GNUCXX)
    #SET(COMPILER_FLAGS "-std=c++11 -stdlib=libc++ -I ${CLANG_INCLUDE} -L $
        {CLANG_LIB}")
ENDIF(CMAKE_COMPILER_IS_GNUCXX)

SET(COMPILER_FLAGS "${COMPILER_FLAGS} -g3 -fvar-tracking -Wall -fno-inline")
SET(COMPILER_FLAGS "${COMPILER_FLAGS} -m64 -Wno-unused-variable -Wno-sign-
    compare -fmax-errors=1 -Werror")
SET(COMPILER_FLAGS "${COMPILER_FLAGS} -Og -fdiagnostics-show-option")
SET(COMPILER_FLAGS "${COMPILER_FLAGS} -Wno-deprecated -Wno-ignored-
    attributes")
# system dlib -L/usr/local/opt/openblas/lib  -I/usr/local/opt/openblas/
    include
# -O0 = optimization is 0 (debug mode) if -O4 (max optimized)
# -g = debug information
# -ferror-limit = 1 -> Max errors shown
# -arch x86_64 = force 64 bit architecture
# -DPLB_MAC_OS_X = Palabos Mac OSX flag
# -Wshorten-64-to-32 = Output warnings when 64bit code is truncated into 32
    bit code
#  Force C++11 --> For CLANG use -std=c++11 -stdlib=libc++ -nostdinc++ FOR
    GNU use -std=gnu++11 -stdlib=libstdc++
# -Wimplicit-function-declaration = Warns if a function is used before it is
    declared (Linking error)
# -Wall = Turns on a lot of usefull warnings
# -ftemplate-backtrace-limit=0 See all backtraces
# -fno-common = Compile common globals like normal definitions
# -fobjc-gc = Enable Objective-C garbage collection
# -mthread-model <value> = The thread model to use, e.g. posix, single
    (posix by default)
# -pthread = Support POSIX threads in generated code

SET(DEFAULT ON CACHE INTERNAL "Default value for enabled by default
    options")
OPTION(ENABLE_MPI "Enable MPI" ${DEFAULT})

IF(APPLE)
    ADD_DEFINITIONS("-DPLB_MAC_OS_X")
    set(COMPILER_FLAGS "-DPLB_MAC_OS_X ${COMPILER_FLAGS}")
```

```
    message("THIS A MAC OSX SYSTEM... DEFINING PLB_MAC_OSX")
    #SET(COMPILER_FLAGS "${COMPILER_FLAGS} -fsanitize=address -fno-omit-
        frame-pointer")
ENDIF(APPLE)

IF(DEFINED "-DPLB_MPI_PARALLEL")
    set(ENABLE_MPI TRUE)
ELSE(DEFINED "-DPLB_MPI_PARALLEL")
    set(ENABLE_MPI FALSE)
ENDIF(DEFINED "-DPLB_MPI_PARALLEL")


IF(ENABLE_MPI)
    include_directories("etc/modulefiles/mpi/openmpi-x86_64")
    INCLUDE(FindMPI)
    IF(MPI_C_FOUND AND MPI_CXX_FOUND)
        SET(CMAKE_C_COMPILER ${MPI_C_COMPILER})
        SET(CMAKE_CXX_COMPILER ${MPI_CXX_COMPILER})
        ADD_DEFINITIONS("-DPLB_MPI_PARALLEL")
        set(COMPILER_FLAGS "-DPLB_MPI_PARALLEL ${COMPILER_FLAGS}")
        message("MPI PARALLEL IS ENABLED... DEFINING PLB_MPI_PARALLEL")
    ELSE(MPI_C_FOUND AND MPI_CXX_FOUND)
        #Check old FindMPI version
        IF(MPI_COMPILER)
            SET(CMAKE_C_COMPILER ${MPI_COMPILER})
            SET(CMAKE_CXX_COMPILER ${MPI_COMPILER})
            ADD_DEFINITIONS("-DPLB_MPI_PARALLEL")
            set(COMPILER_FLAGS "-DPLB_MPI_PARALLEL ${COMPILER_FLAGS}")
            message("MPI PARALLEL IS ENABLED... DEFINING PLB_MPI_PARALLEL")
        include_directories("/usr/lib/openmpi/lib/")
        ELSE(MPI_COMPILER)
            MESSAGE(FATAL_ERROR "MPI-COMPILER NOT found!")
        ENDIF(MPI_COMPILER)
    ENDIF(MPI_C_FOUND AND MPI_CXX_FOUND)
ENDIF(ENABLE_MPI)

OPTION(ENABLE_POSIX "Enable POSIX" ${DEFAULT})

IF(ENABLE_POSIX)
    if(APPLE)
        message("This is an Apple hence disabling -pthread")
    else(APPLE)
        ADD_DEFINITIONS("-DPLB_USE_POSIX ")
        ADD_DEFINITIONS("-pthread ")
        set(COMPILER_FLAGS "-DPLB_USE_POSIX -pthread ${COMPILER_FLAGS}")
        message("POSIX IS ENABLED... DEFINING PLB_USE_POSIX")
    endif(APPLE)
ENDIF(ENABLE_POSIX)

#OPTION(ENABLE_SMP_PARALLEL "Enable SMP_PARALLEL" ${DEFAULT})

IF(ENABLE_SMP_PARALLEL)
    ADD_DEFINITIONS("-DPLB_SMP_PARALLEL")
    ADD_DEFINITIONS("-mthread-model SMP ")
    set(COMPILER_FLAGS "-DPLB_SMP_PARALLEL -mthread-model SMP $
        {COMPILER_FLAGS}")
    message("SMP PARALLEL IS ENABLED... DEFINING PLB_SMP_PARALLEL")
ENDIF(ENABLE_SMP_PARALLEL)

#Set Debug flags
```

```
ADD_DEFINITIONS("-DPLB_DEBUG")
set(COMPILER_FLAGS "-DPLB_DEBUG  ${COMPILER_FLAGS}")

message("COMPILER_FLAGS: " ${COMPILER_FLAGS})
SET(CMAKE_BUILD_TYPE "Debug") #None, Debug, Release
SET(CMAKE_CXX_FLAGS ${COMPILER_FLAGS})
SET(CMAKE_C_FLAGS ${COMPILER_FLAGS})

#============================================================================
    ========================================
#
#       FILES
#
#============================================================================
    ========================================

# Select the executable cpp
set(PROJECT_PATH "/viscosityTest/src/")
set(PROJECT_EXE "test.cpp")

# Make a list for the source files
set(SOURCE "${PALABOS_ROOT}${PROJECT_PATH}${PROJECT_EXE}")
# Make a list for the include directories
set(DIRECTORY_LIST "${PALABOS_ROOT}${PROJECT_PATH}")

#Include the Tinyxml include dir
set(DIRECTORY_LIST ${DIRECTORY_LIST} ${TINYXML_INCLUDE_DIR})


add_subdirectory("${CMAKE_CURRENT_SOURCE_DIR}/src/")



############################################################################
    ######
#
#       LINK AND INSTALL
#
############################################################################
    ######

add_executable(viscosityTest ${SOURCE})
add_sanitizers(viscosityTest)

target_include_directories(viscosityTest PUBLIC
$<BUILD_INTERFACE:${PROJECT_PATH}>
$<INSTALL_INTERFACE:"${CMAKE_INSTALL_PREFIX}/bin/">  # <prefix>/include/
    mylib
)

IF(ENABLE_MPI)
    IF((MPI_C_FOUND AND MPI_CXX_FOUND) OR MPI_COMPILER)
        target_link_libraries(viscosityTest palabos tinyxml $
            {MPI_CXX_LIBRARIES})
    ENDIF((MPI_C_FOUND AND MPI_CXX_FOUND) OR MPI_COMPILER)
ELSE(ENABLE_MPI)
    target_link_libraries(viscosityTest palabos tinyxml)
ENDIF(ENABLE_MPI)
```

CMakeLists.txt 07/12/2016, 13:32

```
# List all header files in palabos/src
# subdirs of palabos/src
set(sub_dirs "dataProcessors" "offLattice" "algorithm" "finiteDifference"
"atomicBlock" "io" "basicDynamics" "latticeBoltzmann"   "boundaryCondition"
    "libraryInterfaces"
"coProcessors" "multiBlock" "parallelism" "complexDynamics" "multiGrid"
    "particles" "core" "multiPhysics" "modules")
# List all header files in the sub directories
foreach(sub ${sub_dirs})
    file(GLOB FOUND_H_FILES "${sub}/*.h")
    if(FOUND_H_FILES)
        set(H_FILES ${H_FILES} ${FOUND_H_FILES})
    endif()
    file(GLOB FOUND_HH_FILES "${sub}/*.hh")
    if(FOUND_HH_FILES)
        set(HH_FILES ${HH_FILES} ${FOUND_HH_FILES})
    endif()
    file(GLOB FOUND_CXX_FILES "${sub}/*.cpp")
    if(FOUND_CXX_FILES)
        set(CXX_FILES ${CXX_FILES} ${FOUND_CXX_FILES})
    endif()
    set(DIRECTORY_LIST ${DIRECTORY_LIST} "${sub}/")
endforeach()

file(GLOB FOUND_H_FILES "*.h")
if(FOUND_H_FILES)
    set(H_FILES ${H_FILES} ${FOUND_H_FILES})
endif()
file(GLOB FOUND_HH_FILES "*.hh")
if(FOUND_HH_FILES)
    set(HH_FILES ${HH_FILES} ${FOUND_HH_FILES})
endif()
file(GLOB FOUND_CXX_FILES "*.cpp")
if(FOUND_CXX_FILES)
    set(CXX_FILES ${CXX_FILES} ${FOUND_CXX_FILES})
endif()

add_library(palabos STATIC ${CXX_FILES} ${H_FILES} ${HH_FILES})

set_target_properties(palabos PROPERTIES LINKER_LANGUAGE CXX)

target_include_directories(palabos PUBLIC
$<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
$<INSTALL_INTERFACE:"${CMAKE_INSTALL_PREFIX}/lib/">  # <prefix>/include/
    mylib
)

target_link_libraries(palabos tinyxml)
```

main.cpp 07/12/2016, 13:34

A.2. Main

```
//
//  test.cpp
//  Palabos
//
//  Created by MAC on 10/02/16.
//  Copyright © 2016 TUDelft. All rights reserved.
//
//
#ifdef PLB_DEBUG
    #define NDEBUG
#endif

typedef double T;

// PALABOS INCLUDES
#include <palabos3D.h>
#include <palabos3D.hh>

// MY INCLUDES
#include "myheaders3D.hh"

//C++ includes
#include <string>

#define Descriptor plb::descriptors::D3Q19Descriptor

typedef plb::Array<T,3> BoundaryType;
typedef plb::Array<T,3> SurfaceData;

int main(int argc, char* argv[])
{
    try{
        //plb::installSigHandler();
        plb::plbInit(&argc, &argv, true); // Initialize Palabos
        bool master = plb::global::mpi().isMainProcessor();
        std::string fileName = "";
        plb::global::argv(argc-1).read(fileName);
        plb::Helper<T,BoundaryType,SurfaceData,Descriptor> h;
        h.initialize(fileName);
        plb::Constants<T>* constants = plb::Constants<T>::c.get();
        plb::Wall<T,BoundaryType,SurfaceData,Descriptor>* wall = plb::Wall<T
            ,BoundaryType,SurfaceData,Descriptor>::w.get();
        plb::Obstacle<T,BoundaryType,SurfaceData,Descriptor>* obstacle =
            plb::Obstacle<T,BoundaryType,SurfaceData,Descriptor>::o.get();
        plb::Variables<T,BoundaryType,SurfaceData,Descriptor>* variables =
            plb::Variables<T,BoundaryType,SurfaceData,Descriptor>::v.get();
        plb::Output<T,BoundaryType,SurfaceData,Descriptor>* output = plb::
            Output<T,BoundaryType,SurfaceData,Descriptor>::out.get();
        output->startMessage();
        output->elapsedTime(); // Initialize Test Timer
        #ifdef PLB_DEBUG
            plb::pcout << "Min Reynolds = "<<constants->minRe<<" Max
                Reynolds = "<<constants->maxRe << std::endl;
            plb::pcout << "Min Grid Level = 0 Max Grid Level = "<<constants-
                >maxGridLevel << std::endl;
            plb::global::profiler().turnOn();
        #endif
        for(plb::plint reynolds = constants->minRe; reynolds <= constants->
            maxRe; reynolds++){
            for(plb::plint gridLevel = 0; gridLevel<= constants->
```

main.cpp 07/12/2016, 13:34

```
                maxGridLevel; gridLevel++){
                variables->update(gridLevel,reynolds);
                variables->setLattice();
                bool converged = false;
                bool stop = false;
                for(int i=0; converged == false; i++)
                {
                    variables->iter++;
                    variables->time = i + 1.0;
                    //variables->lattice->toggleInternalStatistics(true);
                    //obstacle->updateImmersedWall();
                    variables->lattice->executeInternalProcessors(); //
                        Execute all processors and communicate
                        appropriately.
                    variables->lattice->incrementTime();
                    //variables->lattice->collideAndStream();
                    stop = obstacle->move();
                    variables->updateLattice();
                    //if(variables->checkConvergence()){ converged = true;
                        break; }
                    #ifdef PLB_DEBUG
                        std::string mesg="N collisions="+std::to_string
                            (variables->iter);
                        if(master){std::cout << mesg << std::endl;}
                        plb::global::log(mesg);
                        if(master){std::cout<<"Grid Level="+std::to_string
                            (gridLevel);}
                        if(master){std::cout << mesg << std::endl;}
                        plb::global::log(mesg);
                        plb::global::profiler().writeReport();
                    #endif
                    output->writeImages(reynolds,gridLevel,stop);
                    if(stop){break;}
                    if(constants->test){ if(variables->iter > constants->
                        testIter){ output->writeImages(reynolds,gridLevel,
                        true); break; }}
                }
            }
        if(constants->test){ break; }
        }
        plb::global::profiler().turnOff();
        output->stopMessage();
        return 0;
                    // Return Process Completed
    }
    catch(const std::exception& e){plb::exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);  return -1;  }
};
```

```
#ifndef MYHEADERS3D_H
#define MYHEADERS3D_H

#include "mpiDataManager.h"
#include "LBMexceptions.h"
#include "debug.h"
#include "geo.h"
#include "velocity.h"
#include "constants.h"
#include "wall.h"
#include "normal.h"
#include "obstacle.h"
#include "variables.h"
#include "output.h"
#include "helper.h"


#endif // MYHEADERS3D_H
```

myheaders3D.hh 07/12/2016, 13:51

```
#ifndef MYHEADERS3D_HH
#define MYHEADERS3D_HH

#include "myheaders3D.h"
#include "mpiDataManager.hh"
#include "geo.hh"
#include "velocity.hh"
#include "normal.hh"
#include "constants.hh"
#include "wall.hh"
#include "obstacle.hh"
#include "variables.hh"
#include "output.hh"
#include "helper.hh"


#endif // MYHEADERS3D_HH
```

helper.h 07/12/2016, 13:26

A.4. Helper

```
#ifndef HELPER_H
#define HELPER_H

#include <palabos3D.h>
#include "myheaders3D.h"

#include <string>

namespace plb{

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
class Helper{
public:
    static int objCount;
    Helper();
    ~Helper();
    void initialize(const std::string& fileName);
private:
//Classes
    Constants<T> constants; //= Constants<T>();
    Wall<T,BoundaryType,SurfaceData,Descriptor> wall; //=
        Wall<T,BoundaryType,SurfaceData,Descriptor>();
    Obstacle<T,BoundaryType,SurfaceData,Descriptor> obstacle; //=
        Obstacle<T,BoundaryType,SurfaceData,Descriptor>();
    Variables<T,BoundaryType,SurfaceData,Descriptor> variables; //=
        Variables<T,BoundaryType,SurfaceData,Descriptor>();
    Output<T,BoundaryType,SurfaceData,Descriptor> output; //=
        Output<T,BoundaryType,SurfaceData,Descriptor>();
    static bool master;
};

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Helper<T,BoundaryType,SurfaceData,Descriptor>::objCount = 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Helper<T,BoundaryType,SurfaceData,Descriptor>::master = false;

}//namespace plb

#endif //HELPER_H
```

helper.hh 07/12/2016, 13:26

```
#ifndef HELPER_HH
#define HELPER_HH

#include <palabos3D.hh>
#include "myheaders3D.hh"

namespace plb{


    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Helper<T,BoundaryType,SurfaceData,Descriptor>::Helper(){
        master = global::mpi().isMainProcessor();
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Constructing Helper";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        if(objCount == 0){
            if(Constants<T>::objCount == 0)
            {
                constants = Constants<T>();
            }
            if(Wall<T,BoundaryType,SurfaceData,Descriptor>::objCount == 0)
            {
                wall = Wall<T,BoundaryType,SurfaceData,Descriptor>();
            }
            if(Obstacle<T,BoundaryType,SurfaceData,Descriptor>::objCount ==
                0)
            {
                obstacle = Obstacle<T,BoundaryType,SurfaceData,Descriptor>()
                    ;
            }
            if(Variables<T,BoundaryType,SurfaceData,Descriptor>::objCount ==
                0)
            {
                variables = Variables<T,BoundaryType,SurfaceData,Descriptor>
                    ();
            }
            if(Output<T,BoundaryType,SurfaceData,Descriptor>::objCount == 0)
            {
                output = Output<T,BoundaryType,SurfaceData,Descriptor>();
            }
        }
        objCount++;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Helper<T,BoundaryType,SurfaceData,Descriptor>::~Helper(){
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Helper";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        if(Constants<T>::objCount > 0)
        {
            constants.~Constants<T>();
        }
        if(Wall<T,BoundaryType,SurfaceData,Descriptor>::objCount > 0)
```

```
        {
            wall.~Wall<T,BoundaryType,SurfaceData,Descriptor>();
        }
        if(Obstacle<T,BoundaryType,SurfaceData,Descriptor>::objCount > 0)
        {
            obstacle.~Obstacle<T,BoundaryType,SurfaceData,Descriptor>();
        }
        if(Variables<T,BoundaryType,SurfaceData,Descriptor>::objCount > 0)
        {
            variables.~Variables<T,BoundaryType,SurfaceData,Descriptor>();
        }
        if(Output<T,BoundaryType,SurfaceData,Descriptor>::objCount > 0)
        {
            output.~Output<T,BoundaryType,SurfaceData,Descriptor>();
        }
        objCount--;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Helper<T,BoundaryType,SurfaceData,Descriptor>::initialize(const
        std::string& fileName){
        if(Constants<T>::objCount == 1)
        {
            constants.initialize(fileName);
        }
        if(Wall<T,BoundaryType,SurfaceData,Descriptor>::objCount == 1)
        {
            wall.initialize();
        }
        if(Obstacle<T,BoundaryType,SurfaceData,Descriptor>::objCount == 1)
        {
            obstacle.initialize();
        }
        if(Variables<T,BoundaryType,SurfaceData,Descriptor>::objCount == 1)
        {
            variables.initialize();
        }
        if(Output<T,BoundaryType,SurfaceData,Descriptor>::objCount == 1)
        {
            output.initialize();
        }
    }


} // namespace plb

#endif // HELPER_HH
```

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#include <palabos3D.h>
#include "myheaders3D.h"
#include <memory>
#include <string>

namespace plb{

template<typename T>
struct Object{
    std::string fileName;
    Array<T,3> start;
    Array<T,3> dim;
    int referenceDirection;
    double density;
    std::string material;
    double initialTemperature;
    bool dynamicMesh;
};

template<typename T>
struct Param{
    T u;
    T length;
    T resolution;
    T lx;
    T ly;
    T lz;
    T dx;
};

template<typename T>
class Constants{
private:
public:
    static int objCount;

    Constants();

    ~Constants();

    void initialize(const std::string& fileName);

    void createMinimalSettings();

// Properties
    static Object<T> obstacle, wall;
    static Param<T> physical, lb;
    static std::string parameterXmlFileName;
    static plint testIter, ibIter, testRe, testTime, maxRe, minRe,
        maxGridLevel, margin,
        borderWidth, extraLayer, blockSize, envelopeWidth;
    static T initialTemperature, gravitationalAcceleration, epsilon, maxT,
        imageSave;
    static bool test;
    static Precision precision;
    static std::unique_ptr<Constants<T> > c;
private:
```

```
    static bool master;
};

template<typename T>
int Constants<T>::objCount= 0;

template<typename T>
Object<T> Constants<T>::obstacle;

template<typename T>
Object<T> Constants<T>::wall;

template<typename T>
Param<T> Constants<T>::physical;

template<typename T>
Param<T> Constants<T>::lb;

template<typename T>
std::string Constants<T>::parameterXmlFileName= "";

template<typename T>
plint Constants<T>::extraLayer= 0;

template<typename T>
plint Constants<T>::borderWidth= 0;

template<typename T>
plint Constants<T>::maxGridLevel= 0;

template<typename T>
plint Constants<T>::envelopeWidth= 0;

template<typename T>
plint Constants<T>::maxRe= 0;

template<typename T>
plint Constants<T>::minRe= 0;

template<typename T>
plint Constants<T>::testRe= 0;

template<typename T>
plint Constants<T>::margin= 0;

template<typename T>
plint Constants<T>::testIter= 0;

template<typename T>
plint Constants<T>::ibIter= 0;

template<typename T>
plint Constants<T>::testTime= 0;

template<typename T>
plint Constants<T>::blockSize= 0;

template<typename T>
T Constants<T>::maxT= 0;
```

```
template<typename T>
T Constants<T>::initialTemperature= 0;

template<typename T>
T Constants<T>::gravitationalAcceleration= 0;

template<typename T>
T Constants<T>::epsilon= 0;

template<typename T>
T Constants<T>::imageSave= 0;

template<typename T>
bool Constants<T>::test= false;

template<typename T>
bool Constants<T>::master= false;

template<typename T>
Precision Constants<T>::precision = FLT;

template<typename T>
std::unique_ptr<Constants<T> > Constants<T>::c(nullptr);

}//namespace plb

#endif //CONSTANTS_H
```

constants.hh 07/12/2016, 13:25

```
#ifndef CONSTANTS_HH
#define CONSTANTS_HH

#include "constants.h"
#include <palabos3D.hh>
#include "myheaders3D.hh"

namespace plb{

    template<typename T>
    Constants<T>::Constants()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Constants";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            this->parameterXmlFileName = ""; this->epsilon=0; this->minRe =
                0; this->maxRe=0;this->maxGridLevel=0;
            this->margin=0; this->borderWidth=0; this->extraLayer=0; this->
                blockSize=0; this->envelopeWidth=0; this->initialTemperature
                =0;
            this->gravitationalAcceleration=0; this->master = false;
            this->test = true; this->testRe = 0; this->testTime = 20; this->
                maxT = 0; this->imageSave = 0; this->testIter = 0;
            this->master = global::mpi().isMainProcessor();
            this->c.reset(this);
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Constants already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T>
    Constants<T>::~Constants()
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Constants";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        objCount--;
    }

    template<typename T>
    void Constants<T>::initialize(const std::string& fileName)
    {
        try{
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Creating Constants";
```

```
    if(this->master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif
this->parameterXmlFileName = fileName;
XMLreader r(fileName);
std::string dir;
r["dir"]["out"].read(dir);
global::directories().setOutputDir(dir);
r["dir"]["log"].read(dir);
global::directories().setLogOutDir(dir);
r["dir"]["image"].read(dir);
global::directories().setImageOutDir(dir);

r["lbm"]["maxRe"].read(this->maxRe);
r["lbm"]["minRe"].read(this->minRe);
r["lbm"]["epsilon"].read(this->epsilon);
r["lbm"]["gravity"].read(this->gravitationalAcceleration);
r["lbm"]["u0lb"].read(lb.u);
r["lbm"]["u0"].read(physical.u);
r["lbm"]["pl"].read(physical.resolution);

r["refinement"]["margin"].read(this->margin);
r["refinement"]["borderWidth"].read(this->borderWidth);
if(this->margin < this->borderWidth){throw marginEx;} //
    Requirement: margin>=borderWidth.
r["refinement"]["maxGridLevel"].read(this->maxGridLevel);
r["refinement"]["extraLayer"].read(this->extraLayer);
r["refinement"]["blockSize"].read(this->blockSize);
r["refinement"]["envelopeWidth"].read(this->envelopeWidth);

std::string val;
r["wall"]["meshFileName"].read(val);
wall.fileName = val;
r["wall"]["dynamicMesh"].read(wall.dynamicMesh);
r["wall"]["material"].read(wall.material);
r["wall"]["referenceDirection"].read(wall.referenceDirection);
r["wall"]["initialTemperature"].read(wall.initialTemperature);
Array<T,3> dim = Array<T,3>(0,0,0);
r["wall"]["lx"].read(dim[0]);
r["wall"]["ly"].read(dim[1]);
r["wall"]["lz"].read(dim[2]);
wall.dim = dim;

r["obstacle"]["meshFileName"].read(val);
obstacle.fileName = val;
r["obstacle"]["dynamicMesh"].read(obstacle.dynamicMesh);
r["obstacle"]["material"].read(obstacle.material);
Array<T,3> start = Array<T,3>(0,0,0);
r["obstacle"]["obstacleStartX"].read(start[0]);
r["obstacle"]["obstacleStartY"].read(start[1]);
r["obstacle"]["obstacleStartZ"].read(start[2]);
obstacle.start = start;
r["obstacle"]["referenceDirection"].read(obstacle.
    referenceDirection);
r["obstacle"]["density"].read(obstacle.density);
dim = Array<T,3>(0,0,0);
r["obstacle"]["lx"].read(dim[0]);
r["obstacle"]["ly"].read(dim[1]);
r["obstacle"]["lz"].read(dim[2]);
obstacle.dim = dim;
```

```
T l = 1000;
for(int i = 0; i<3; i++){
    if(wall.dim[i] < l){ l = wall.dim[i]; }
    if(obstacle.dim[i] < l){ l = obstacle.dim[i]; }
}
physical.length = l;
lb.dx = physical.length / physical.resolution;
lb.lx = wall.dim[0] / lb.dx;
physical.lx = wall.dim[0];
lb.ly = wall.dim[1] / lb.dx;
physical.ly = wall.dim[1];
lb.lz = wall.dim[2] / lb.dx;
physical.lz = wall.dim[2];
//wall.dim = wall.dim / dx;
//obstacle.dim = obstacle.dim / dx;

int tmp = 0;
r["simulation"]["test"].read(tmp);
if(tmp == 1){ this->test = true; }else{ this->test = false;}
r["simulation"]["testRe"].read(this->testRe);
r["simulation"]["testTime"].read(this->testTime);
r["simulation"]["maxT"].read(this->maxT);
r["simulation"]["imageSave"].read(this->imageSave);
r["simulation"]["testIter"].read(this->testIter);
r["simulation"]["ibIter"].read(this->ibIter);
int prec = 0;
r["simulation"]["precision"].read(prec);
r["simulation"]["initialTemperature"].read(this->
    initialTemperature);
switch(prec){
    case 1: this->precision = Precision::FLT;
    case 2: this->precision = Precision::DBL;
    case 3: this->precision = Precision::LDBL;
}
double ratio = 3;
double x = 1;
double y = 3;
double cs = sqrt(x / y);
double maxMach = 0.1;
// Fill the 2D array with standard values
for(plint grid = 0; grid <= this->maxGridLevel; grid++){
    try{
        T resolution = physical.resolution * util::
            twoToThePowerPlint(grid);
        T scaled_u0lb = lb.u; // util::twoToThePowerPlint(grid);
        double mach = scaled_u0lb / cs;
        if(mach > maxMach){std::cout<<"Local Mach= "<<mach<<"\n"
            ; throw localMachEx;}
        if(resolution == 0){throw resolEx;}
        if(this->test){
            this->minRe = this->testRe; this->maxRe = this->
                testRe+1;
            IncomprFlowParam<T> p = IncomprFlowParam<T>(physical
                .u,scaled_u0lb,testRe,physical.length,resolution
                ,lb.lx,lb.ly,lb.lz);
            // Check local speed of sound constraint
            T dt = p.getDeltaT();
            T dx = p.getDeltaX();
            if(dt > (dx / sqrt(ratio)))
                {std::cout<<"dt:"<<dt<<"<(dx:"<<dx<<"/sqrt("<<
```

```
                            ratio<<")"<<"\n"; throw superEx;}
                    }
                    else{
                        for(int reynolds = 0; reynolds <= this->maxRe;
                            reynolds++){
                            IncomprFlowParam<T> p =
                                IncomprFlowParam<T>(physical.u,scaled_u0lb,
                                    reynolds,physical.length,resolution,lb.
                                    lx,lb.ly,lb.lz);
                            // Check local speed of sound constraint
                            T dt = p.getDeltaT();
                            T dx = p.getDeltaX();
                            if(dt > (dx / sqrt(ratio))){std::cout<<"dt:"<<dt
                                <<"<(dx:"<<dx<<"/sqrt("<<ratio<<")"<<"\n";
                                throw superEx;}
                        }
                    }
                }
                catch(const std::exception& e){
                    if(grid == 0){ throw e; }
                    else{ this->maxGridLevel = grid - 1; break; }
                }
            }
            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Done Creating Constants";
                if(this->master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            objCount++;
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T>
    void Constants<T>::createMinimalSettings(){
        try{
            double ratio = 3;
            double x = 1;
            double y = 3;
            double cs = sqrt(x / y);
            double maxMach = 0.1;
            double minU = lb.u;
            double maxU = maxMach * cs;
            for(int i = 0; i < 3; i++){
                T resolution = physical.resolution * util::
                    twoToThePowerPlint(i);
            }
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

} // namespace plb

#endif // CONSTANTS_HH
```

obstacle.h 07/12/2016, 13:29

A.6. Obstacle

```
#ifndef OBSTACLE_H
#define OBSTACLE_H

#include <palabos3D.h>
#include "myheaders3D.h"
#include <memory>

namespace plb{

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
class Obstacle{
private:
public:
    static int objCount;

    Obstacle();
    // Default Destructor

    ~Obstacle();
// Methods
    void initialize();

    static Array<T,3> getCenter(const ConnectedTriangleSet<T>& triangles);

    static Array<T,3> getCenter();

    static Box3D getDomain();

    static Box3D getDomain(const ConnectedTriangleSet<T>& triangles);

    static Box3D getDomain(const TriangleSet<T>& triangles);

    static T getVolume(const ConnectedTriangleSet<T>& triangles);

    // Function to Move Obstacle to it's starting position
    static void moveToStart();

    static void updateImmersedWall();

    // Function to Move the Obstacle through the Fluid
    bool move();

// Attributes
    static bool firstMove;
    static int flowType;
    static T mass, volume, numTriangles, numVertices, g;
    static std::vector<Array<T,3> > vertices;
    static std::vector<Array<T,3> > unitNormals;
    static std::vector<T> areas;
    static Point<T> position;
    static Array<T,3> rotation, velocity, rotationalVelocity, acceleration,
        rotationalAcceleration, location;
    static ConnectedTriangleSet<T> triangleSet;
    static std::unique_ptr<TriangleBoundary3D<T> > tb;
    static std::unique_ptr<VoxelizedDomain3D<T> > vd;
    static std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > lattice;
    static std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > bp;
    static std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > fs;
    static std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > model;
```

```
    static std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,
        BoundaryType> > bc;
    static std::unique_ptr<Obstacle<T,BoundaryType,SurfaceData,Descriptor> >
        o;
    static SurfaceVelocity<T> velocityFunc;
private:
    static SurfaceNormal<T> normalFunc;
    static Array<T,3> rotation_LB, velocity_LB, rotationalVelocity_LB,
        acceleration_LB, rotationalAcceleration_LB, location_LB;
    static bool master;
};

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Obstacle<T,BoundaryType,SurfaceData,Descriptor>::objCount= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Obstacle<T,BoundaryType,SurfaceData,Descriptor>::firstMove= true;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Obstacle<T,BoundaryType,SurfaceData,Descriptor>::flowType= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::mass= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::volume= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::numTriangles= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::numVertices= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::g= 9.81;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<Array<T,3> > Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    vertices;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<Array<T,3> > Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    unitNormals;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<T> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::areas;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
```

```
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::acceleration =
    Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::rotation = Array
    <T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    rotationalVelocity = Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    rotationalAcceleration = Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::velocity = Array
    <T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::location = Array
    <T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Obstacle<T,BoundaryType,SurfaceData,Descriptor>::master = false;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::acceleration_LB
    = Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::rotation_LB =
    Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    rotationalVelocity_LB = Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    rotationalAcceleration_LB = Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::velocity_LB =
    Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::location_LB =
```

```
    Array<T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Point<T> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::position = Point<T
    >(0,0,0);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
ConnectedTriangleSet<T> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    triangleSet = ConnectedTriangleSet<T>(TriangleSet<T>(FLT));

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<TriangleBoundary3D<T> > Obstacle<T,BoundaryType,SurfaceData,
    Descriptor>::tb(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<VoxelizedDomain3D<T> > Obstacle<T,BoundaryType,SurfaceData,
    Descriptor>::vd(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > Obstacle<T,BoundaryType,
    SurfaceData,Descriptor>::lattice(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > Obstacle<T,BoundaryType,
    SurfaceData,Descriptor>::bp(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > Obstacle<T,BoundaryType
    ,SurfaceData,Descriptor>::fs(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > Obstacle<T,BoundaryType
    ,SurfaceData,Descriptor>::model(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,BoundaryType> >
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::bc(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
SurfaceVelocity<T> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
    velocityFunc = SurfaceVelocity<T>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
SurfaceNormal<T> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::normalFunc
    = SurfaceNormal<T>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<Obstacle<T,BoundaryType,SurfaceData,Descriptor> > Obstacle<T
```

```
    ,BoundaryType,SurfaceData,Descriptor>::o(nullptr);

}// namespace plb

#endif //OBSTACLE_H
```

obstacle.hh 07/12/2016, 13:29

```
#ifndef OBSTACLE_HH
#define OBSTACLE_HH

#include "obstacle.h"
#include <palabos3D.hh>
#include "myheaders3D.hh"
#include <string>
#include <exception>

namespace plb{

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::Obstacle()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Obstacle";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            this->o.reset(this);
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Obstacle already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::~Obstacle()
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Obstacle";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        objCount--;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Obstacle<T,BoundaryType,SurfaceData,Descriptor>::initialize()
    {
        try{
            std::string meshFileName = Constants<T>::obstacle.fileName;
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Initializing Obstacle";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg ="[DEBUG] MeshFileName ="+meshFileName;
```

```
    if(master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif

velocity[0] = 0;    velocity[1] = 0;    velocity[2] = 0;
acceleration[0] = 0;    acceleration[1] = 0;    acceleration[2]
    = 0;
rotation[0] = 0;    rotation[1] = 0; rotation[2] = 0;
rotationalVelocity[0] = 0;  rotationalVelocity[1] = 0;
    rotationalVelocity[2] = 0;
rotationalAcceleration[0] = 0;  rotationalAcceleration[1] = 0;
    rotationalAcceleration[2] = 0;
TriangleSet<T> surface;
#ifdef PLB_MPI_PARALLEL
    if(global::mpi().isMainProcessor()){
        surface = TriangleSet<T>(meshFileName, Constants<T>::
            precision, STL);
        global::mpiData().sendTriangleSet<T>(surface);
    }
    else{ surface = global::mpiData().receiveTriangleSet<T>(); }
#else
    surface = TriangleSet<T>(meshFileName, Constants<T>::
        precision, STL);
#endif

Box3D domain = getDomain(surface);

#ifdef PLB_DEBUG
    mesg ="[DEBUG] Domain BEFORE Scaling "+ box_string(domain)
        +" in physical units";
    if(master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif

T x = 0;
T y = 0;
T z = 0;
if(domain.x0<0){ x = -domain.x0;}
if(domain.y0<0){ y = -domain.y0;}
if(domain.z0<0){ z = -domain.z0;}
Array<T,3> shift = Array<T,3>(x,y,z);
surface.translate(shift);

T alpha = 0.01;
surface.scale(alpha);

domain = getDomain(surface);

#ifdef PLB_DEBUG
    mesg = "[DEBUG] Domain AFTER scaling "+ box_string(domain)
        +" in physical units";
    if(master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif

const T dx = Constants<T>::lb.dx;

T maxEdgeLength = surface.getMaxEdgeLength();

triangleSet = ConnectedTriangleSet<T>(surface);
```

```
        numVertices = triangleSet.getNumVertices();
        numTriangles = triangleSet.getNumTriangles();

        pcout << "The immersed surface  has " << numVertices << "
            vertices and " << numTriangles << " triangles." << std::endl
            ;
        pcout << "The immersed surface has a maximum triangle edge
            length of " << maxEdgeLength << std::endl;
        pcout << "dx = "<<dx <<std::endl;

        if (maxEdgeLength >= 4.0 * dx) {
            pcout << std::endl;
            pcout << "CAUTION: The maximum triangle edge length for the
                immersed surface is greater than "
                  << " 4 times dx."
                  << std::endl;
            pcout << "         The immersed boundary method will not
                work correctly. Surface refinement is necessary."
                  << std::endl;
            pcout << std::endl;
            exit(1);
        } else if (maxEdgeLength > dx) {
            pcout << std::endl;
            pcout << "WARNING: The maximum triangle edge length for the
                immersed surface is greater than dx."
                  << std::endl;
            pcout << "         The immersed boundary method might not
                work in an optimal way. Surface refinement is
                recommended."
                  << std::endl;
            pcout << std::endl;
        }

        for (plint iVertex = 0; iVertex < numVertices; iVertex++) {
            vertices.push_back(triangleSet.getVertex(iVertex));
            T area;
            Array<T,3> unitNormal;
            triangleSet.computeVertexAreaAndUnitNormal(iVertex, area,
                unitNormal);
            areas.push_back(area);
            unitNormals.push_back(unitNormal);
        }

        flowType = voxelFlag::outside;
        volume = getVolume(triangleSet);
        mass = Constants<T>::obstacle.density * volume;
        g = Constants<T>::gravitationalAcceleration;
        velocityFunc.initialize(mass, g, Constants<T>::obstacle.density)
            ;

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Volume= "+std::to_string(volume)+" Mass= "+
                std::to_string(mass);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Done Initializing Obstacle";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
```

```
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}


template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getCenter
    (const ConnectedTriangleSet<T>& triangles)
{
    Array<T,3> cg = Array<T,3>(0,0,0);
    try{
        #ifdef PLB_DEBUG
            std::string mesg ="[DEBUG] Calculating Center";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        T x = 0;
        T y = 0;
        T z = 0;
        numVertices = triangles.getNumVertices();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = triangles.getVertex(i);
            x += iVertex[0];
            y += iVertex[1];
            z += iVertex[2];
        }
        cg = Array<T,3>(x/numVertices, y/numVertices, z/numVertices);
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] DONE Center= "+array_string(cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return cg;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Array<T,3> Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getCenter()
{
    Array<T,3> cg = Array<T,3>(0,0,0);
    try{
        #ifdef PLB_DEBUG
            std::string mesg ="[DEBUG] Calculating Center";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        T x = 0;
        T y = 0;
        T z = 0;
        numVertices = tb->getMesh().getNumVertices();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = tb->getMesh().getVertex(i);
            x += iVertex[0];
            y += iVertex[1];
            z += iVertex[2];
        }
```

```
        cg = Array<T,3>(x/numVertices, y/numVertices, z/numVertices);
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] DONE Center= "+array_string(cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return cg;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Box3D Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getDomain(const
    ConnectedTriangleSet<T>& triangles)
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        T numVertices = triangles.getNumVertices();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = triangles.getVertex(i);
            if(iVertex[0] < xmin){ xmin = iVertex[0]; }
            if(iVertex[0] > xmax){ xmax = iVertex[0]; }
            if(iVertex[1] < ymin){ ymin = iVertex[1]; }
            if(iVertex[1] > ymax){ ymax = iVertex[1]; }
            if(iVertex[2] < zmin){ zmin = iVertex[2]; }
            if(iVertex[2] > zmax){ zmax = iVertex[2]; }
        }
        d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return d;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Box3D Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getDomain(const
    TriangleSet<T>& triangles)
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        std::vector<Array<Array<T,3>,3> >  iTriangles = triangles.
            getTriangles();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<iTriangles.size(); i++){
```

```
            Array<Array<T,3>,3> iTriangle = iTriangles[i];
            for(int v = 0; v<3; v++){
                Array<T,3> iVertex = iTriangle[v];
                if(iVertex[0] < xmin){ xmin = iVertex[0]; }
                if(iVertex[0] > xmax){ xmax = iVertex[0]; }
                if(iVertex[1] < ymin){ ymin = iVertex[1]; }
                if(iVertex[1] > ymax){ ymax = iVertex[1]; }
                if(iVertex[2] < zmin){ zmin = iVertex[2]; }
                if(iVertex[2] > zmax){ zmax = iVertex[2]; }
            }
        }
        d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return d;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Box3D Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getDomain()
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        T numVertices = tb->getMesh().getNumVertices();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = tb->getMesh().getVertex(i);
            if(iVertex[0] < xmin){ xmin = iVertex[0]; }
            if(iVertex[0] > xmax){ xmax = iVertex[0]; }
            if(iVertex[1] < ymin){ ymin = iVertex[1]; }
            if(iVertex[1] > ymax){ ymax = iVertex[1]; }
            if(iVertex[2] < zmin){ zmin = iVertex[2]; }
            if(iVertex[2] > zmax){ zmax = iVertex[2]; }
        }
        d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return d;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
T Obstacle<T,BoundaryType,SurfaceData,Descriptor>::getVolume(const
    ConnectedTriangleSet<T>& triangles)
{
    T v = 0;
    try{
        Array<T,3> cg = getCenter(triangles);
        #ifdef PLB_DEBUG
            std::string mesg ="[DEBUG] Calculating Volume";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
```

```
        #endif
        numTriangles = triangles.getNumTriangles();
        for(int i =0; i<numTriangles; i++){
            Array<T,3> iTriangle = triangles.getTriangle(i);
            Array<T,3> a = triangleSet.getVertex(iTriangle[0]);
            Array<T,3> b = triangleSet.getVertex(iTriangle[1]);
            Array<T,3> c = triangleSet.getVertex(iTriangle[2]);
            T iVolume = computeTetrahedronSignedVolume(a,b,c,cg);
            if(iVolume<0){iVolume *= -1; }
            v += iVolume;
        }
        if(v<0){
            std::string ex = "[ERROR] Volume= "+std::to_string(v)+" is
                negative!";
            throw std::runtime_error(ex);
        }
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] DONE Volume= "+std::to_string(v);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return v;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Obstacle<T,BoundaryType,SurfaceData,Descriptor>::moveToStart()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Moving Obstacle to Start
                Position";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("obstacle").start();
        #endif
            const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor
                >::p.getDeltaX();

            Box3D wall_domain = Wall<T,BoundaryType,SurfaceData,
                Descriptor>::getDomain();
            Array<T,3> wall_cg = Wall<T,BoundaryType,SurfaceData,
                Descriptor>::getCenter();
            // Find the current location
            Box3D obstacle_domain = getDomain();
            Array<T,3> obstacle_cg = getCenter();

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Obstacle Original Position= "+box_string
                (obstacle_domain)+" Center= "+array_string(obstacle_cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
            T x = 0;
            T y = 0;
            T z = 0;
            x = wall_cg[0] - obstacle_cg[0];
```

```
            y = wall_cg[1] - obstacle_cg[1];
            z = wall_domain.z1 - obstacle_domain.z1-1;

            tb->getMesh().translate(Array<T,3>(x,y,z));

            unitNormals.clear();
            unitNormals.resize(numVertices);
            unitNormals.reserve(numVertices);

            areas.clear();
            areas.resize(numVertices);
            areas.reserve(numVertices);

            for(int i = 0; i < numVertices; i++){
                areas[i] = tb->getMesh().computeVertexArea(i);
                unitNormals[i] = tb->getMesh().computeVertexNormal(i);
            }

            obstacle_domain = getDomain();
            obstacle_cg = getCenter();

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Wall Domain= "+box_string(wall_domain)+"
                Center= "+array_string(wall_cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg = "[DEBUG] Obstacle Start Position= "+box_string
                (obstacle_domain)+" Center= "+array_string(obstacle_cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg = "[DEBUG] DONE Moving Obstacle to Start Position";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("obstacle").stop();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Obstacle<T,BoundaryType,SurfaceData,Descriptor>::updateImmersedWall
    ()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Updating Immersed Wall";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("update").start();
        #endif

            numVertices = tb->getMesh().getNumVertices();
            if(numVertices == 0){throw std::runtime_error("No vertices
                returned from obstacle TriangleBoundary"); }
            vertices.clear();
            vertices.resize(numVertices);
            vertices.reserve(numVertices);
```

```
            unitNormals.clear();
            unitNormals.resize(numVertices);
            unitNormals.reserve(numVertices);

            areas.clear();
            areas.resize(numVertices);
            areas.reserve(numVertices);

            const bool weightedArea = false;

            for(int i = 0; i < numVertices; i++){
                vertices[i] = tb->getMesh().getVertex(i);
                areas[i] = tb->getMesh().computeVertexArea(i);
                unitNormals[i] = tb->getMesh().computeVertexNormal(i,
                    weightedArea);
            }

            //instantiateImmersedWallData(vertices, areas,
                *Variables<T,BoundaryType,SurfaceData,Descriptor>::conta
                iner);

            std::vector<MultiBlock3D*> args;
            plint pl = 4;

            args.resize(0);
            args.push_back(Variables<T,BoundaryType,SurfaceData,
                Descriptor>::container);
            integrateProcessingFunctional(new
                InstantiateImmersedWallData3D<T>(
                vertices,
                areas,
                unitNormals),
                Variables<T,BoundaryType,SurfaceData,Descriptor>::
                    container->getBoundingBox(),
                *Variables<T,BoundaryType,SurfaceData,Descriptor>::
                    lattice, args, pl);

        #ifdef PLB_DEBUG
            mesg =   "[DEBUG] DONE Updating Immersed Wall";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("update").stop();
        #endif
        }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
bool Obstacle<T,BoundaryType,SurfaceData,Descriptor>::move()
{
    bool stop = false;
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Moving Obstacle";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("move").start();
        #endif
```

```
const T dt = Variables<T,BoundaryType,SurfaceData,Descriptor
    >::p.getDeltaT();
const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor
    >::p.getDeltaX();
const T omega = Variables<T,BoundaryType,SurfaceData,
    Descriptor>::p.getOmega();
const T rho_LB = (T)1.0;
const T timeLB = Variables<T,BoundaryType,SurfaceData,
    Descriptor>::time;
const Box3D lattice_domain = Variables<T,BoundaryType,
    SurfaceData,Descriptor>::lattice->getBoundingBox();
const Box3D obstacle_domain = getDomain();
normalFunc.update(tb.get());

T factor = util::sqr(util::sqr(dx)) / util::sqr(dt);

resetForceStatistics<T>(*Variables<T,BoundaryType,
    SurfaceData,Descriptor>::container);

recomputeImmersedForce<T>(normalFunc, omega, rho_LB,
    *Variables<T,BoundaryType,SurfaceData,Descriptor>::
        lattice,
    *Variables<T,BoundaryType,SurfaceData,Descriptor>::
        container,
    Constants<T>::envelopeWidth, obstacle_domain, true);

Array<T,3> force = Array<T,3>(0,0,0);
force = -reduceImmersedForce<T>(*Variables<T,BoundaryType,
    SurfaceData,Descriptor>::container, voxelFlag::outside);

Array<T,3> center = getCenter();

Array<T,3> torque = Array<T,3>(0,0,0);
torque = -reduceAxialTorqueImmersed(*Variables<T,
    BoundaryType,SurfaceData,Descriptor>::container,
                        center, Array<T,3>(1,1,1),
                            voxelFlag::outside);

stop = velocityFunc.update(Variables<T,BoundaryType,
    SurfaceData,Descriptor>::p,
                            timeLB,force,torque,tb.get(),
                                lattice_domain);
/*
for (int i = 0; i < Constants<T>::ibIter; i++){
    indexedInamuroIteration<T>(velocityFunc,

                        *Variables<T,BoundaryType,SurfaceDat
                        a,Descriptor>::rhoBar,

                        *Variables<T,BoundaryType,SurfaceDat
                        a,Descriptor>::j,

                        *Variables<T,BoundaryType,SurfaceDat
                        a,Descriptor>::container,

                        Variables<T,BoundaryType,SurfaceData
                        ,Descriptor>::p.getTau(),
                    true);
}*/
```

```
                normalFunc.update(tb.get());

                updateImmersedWall();

            #ifdef PLB_DEBUG
                mesg =   "[DEBUG] DONE Moving Obstacle";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                global::timer("move").stop();
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return stop;
    }


} // namespace plb

#endif //OBSTACLE_HH
```

wall.h 07/12/2016, 13:31

A.7. Wall

```
#ifndef WALL_H
#define WALL_H

#include <palabos3D.h>
#include "myheaders3D.h"
#include <memory>

namespace plb{

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
class Wall{
private:
public:
    static int objCount;

    Wall();

    ~Wall();
// Methods
    static void initialize();

    static Array<T,3> getCenter();

    static Box3D getDomain();

    static Box3D getDomain(const ConnectedTriangleSet<T>& triangles);

    static Box3D getDomain(const TriangleSet<T>& triangles);
// Attributes
    static int flowType;
    static ConnectedTriangleSet<T> triangleSet;
    static Array<T,3> location, center;
    static Box3D domain;
    static std::unique_ptr<TriangleBoundary3D<T> > tb;
    static std::unique_ptr<VoxelizedDomain3D<T> > vd;
    static std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > lattice;
    static std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > bp;
    static std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > fs;
    static std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > model;
    static std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,
        BoundaryType> > bc;
    static std::unique_ptr<Wall<T,BoundaryType,SurfaceData,Descriptor> > w;
private:
    static bool master;
};

// Initializers
template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Wall<T,BoundaryType,SurfaceData,Descriptor>::objCount= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Wall<T,BoundaryType,SurfaceData,Descriptor>::flowType= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
ConnectedTriangleSet<T> Wall<T,BoundaryType,SurfaceData,Descriptor>::
    triangleSet = ConnectedTriangleSet<T>(TriangleSet<T>(FLT));
```

```
template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Wall<T,BoundaryType,SurfaceData,Descriptor>::location = Array<T,3
    >();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Wall<T,BoundaryType,SurfaceData,Descriptor>::center = Array<T,3>
    ();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Box3D Wall<T,BoundaryType,SurfaceData,Descriptor>::domain = Box3D(0,0,0,0,0,
    0);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<TriangleBoundary3D<T> > Wall<T,BoundaryType,SurfaceData,
    Descriptor>::tb(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<VoxelizedDomain3D<T> > Wall<T,BoundaryType,SurfaceData,
    Descriptor>::vd(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > Wall<T,BoundaryType,
    SurfaceData,Descriptor>::lattice(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > Wall<T,BoundaryType,
    SurfaceData,Descriptor>::bp(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > Wall<T,BoundaryType,
    SurfaceData,Descriptor>::fs(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > Wall<T,BoundaryType,
    SurfaceData,Descriptor>::model(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,BoundaryType> >
    Wall<T,BoundaryType,SurfaceData,Descriptor>::bc(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<Wall<T,BoundaryType,SurfaceData,Descriptor> > Wall<T,
    BoundaryType,SurfaceData,Descriptor>::w(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Wall<T,BoundaryType,SurfaceData,Descriptor>::master= false;
}// namespace plb
```

```
#endif //WALL_H
```

wall.hh 07/12/2016, 13:31

```
#ifndef WALL_HH
#define WALL_HH

#include "wall.h"
#include <palabos3D.hh>
#include "myheaders3D.hh"

namespace plb{

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Wall<T,BoundaryType,SurfaceData,Descriptor>::Wall()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Wall";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            this->w.reset(this);
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Wall already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Wall<T,BoundaryType,SurfaceData,Descriptor>::~Wall()
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Wall";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        objCount--;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Wall<T,BoundaryType,SurfaceData,Descriptor>::initialize()
    {
        try
        {
            std::string meshFileName = Constants<T>::wall.fileName;
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Creating Wall";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg ="[DEBUG] MeshFileName ="+meshFileName;
                if(master){std::cout << mesg << std::endl;}
```

```
    global::log(mesg);
#endif
TriangleSet<T> surface;
#ifdef PLB_MPI_PARALLEL
    if(global::mpi().isMainProcessor()){
        surface = TriangleSet<T>(meshFileName, Constants<T>::
            precision, STL);
        global::mpiData().sendTriangleSet<T>(surface);
    }
    else{ surface = global::mpiData().receiveTriangleSet<T>(); }
#else
    surface = TriangleSet<T>(meshFileName, Constants<T>::
        precision, STL);
#endif

Box3D domain = getDomain(surface);

#ifdef PLB_DEBUG
    mesg ="[DEBUG] Domain BEFORE Scaling "+ box_string(domain)
        +" in physical units";
    if(master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif

T x = 0;
T y = 0;
T z = 0;
if(domain.x0<0){ x = -domain.x0;}
if(domain.y0<0){ y = -domain.y0;}
if(domain.z0<0){ z = -domain.z0;}
Array<T,3> shift = Array<T,3>(x,y,z);
surface.translate(shift);

T alpha = 0.01;
surface.scale(alpha);

domain = getDomain(surface);

#ifdef PLB_DEBUG
    mesg = "[DEBUG] Domain AFTER scaling "+ box_string(domain)
        +" in physical units";
    if(master){std::cout << mesg << std::endl;}
    global::log(mesg);
#endif

const T dx = Constants<T>::lb.dx;

T maxEdgeLength = surface.getMaxEdgeLength();

triangleSet = ConnectedTriangleSet<T>(surface);
plint numVertices = triangleSet.getNumVertices();
plint numTriangles = triangleSet.getNumTriangles();

pcout << "The wall surface  has " << numVertices << " vertices
    and " << numTriangles << " triangles." << std::endl;
pcout << "The wall surface has a maximum triangle edge length of
    " << maxEdgeLength << std::endl;
pcout << "dx = "<<dx <<std::endl;

if (maxEdgeLength >= 4.0 * dx) {
```

```
            pcout << std::endl;
            pcout << "CAUTION: The maximum triangle edge length for the
                immersed surface is greater than "
                  << " 4 times dx."
                  << std::endl;
            pcout << "         The immersed boundary method will not
                work correctly. Surface refinement is necessary."
                  << std::endl;
            pcout << std::endl;
            exit(1);
        } else if (maxEdgeLength > dx) {
            pcout << std::endl;
            pcout << "WARNING: The maximum triangle edge length for the
                immersed surface is greater than dx."
                  << std::endl;
            pcout << "         The immersed boundary method might not
                work in an optimal way. Surface refinement is
                recommended."
                  << std::endl;
            pcout << std::endl;
        }
        flowType = voxelFlag::inside;
        domain = getDomain(triangleSet);
        #ifdef PLB_DEBUG
            mesg="[DEBUG] Done Initializing Wall";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Array<T,3> Wall<T,BoundaryType,SurfaceData,Descriptor>::getCenter()
{
    Array<T,3> cg = Array<T,3>(0,0,0);
    try{
        #ifdef PLB_DEBUG
            std::string mesg ="[DEBUG] Calculating Center";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        T x = 0;
        T y = 0;
        T z = 0;
        T numVertices = tb->getMesh().getNumVertices();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = tb->getMesh().getVertex(i);
            x += iVertex[0];
            y += iVertex[1];
            z += iVertex[2];
        }
        cg = Array<T,3>(x/numVertices, y/numVertices, z/numVertices);
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] DONE Center= "+array_string(cg);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
```

```
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return cg;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Box3D Wall<T,BoundaryType,SurfaceData,Descriptor>::getDomain()
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        T numVertices = tb->getMesh().getNumVertices();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = tb->getMesh().getVertex(i);
            if(iVertex[0] < xmin){ xmin = iVertex[0]; }
            if(iVertex[0] > xmax){ xmax = iVertex[0]; }
            if(iVertex[1] < ymin){ ymin = iVertex[1]; }
            if(iVertex[1] > ymax){ ymax = iVertex[1]; }
            if(iVertex[2] < zmin){ zmin = iVertex[2]; }
            if(iVertex[2] > zmax){ zmax = iVertex[2]; }
        }
        d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return d;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
Box3D Wall<T,BoundaryType,SurfaceData,Descriptor>::getDomain(const
    ConnectedTriangleSet<T>& triangles)
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        T numVertices = triangles.getNumVertices();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = triangles.getVertex(i);
            if(iVertex[0] < xmin){ xmin = iVertex[0]; }
            if(iVertex[0] > xmax){ xmax = iVertex[0]; }
            if(iVertex[1] < ymin){ ymin = iVertex[1]; }
            if(iVertex[1] > ymax){ ymax = iVertex[1]; }
            if(iVertex[2] < zmin){ zmin = iVertex[2]; }
            if(iVertex[2] > zmax){ zmax = iVertex[2]; }
        }
```

```
            d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return d;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Box3D Wall<T,BoundaryType,SurfaceData,Descriptor>::getDomain(const
        TriangleSet<T>& triangles)
    {
        Box3D d(0,0,0,0,0,0);
        try
        {
            std::vector<Array<Array<T,3>,3> > iTriangles = triangles.
                getTriangles();
            T zmin = std::numeric_limits<T>::max();
            T zmax = std::numeric_limits<T>::min();
            T ymin = std::numeric_limits<T>::max();
            T ymax = std::numeric_limits<T>::min();
            T xmin = std::numeric_limits<T>::max();
            T xmax = std::numeric_limits<T>::min();
            for(int i = 0; i<iTriangles.size(); i++){
                Array<Array<T,3>,3> iTriangle = iTriangles[i];
                for(int v = 0; v<3; v++){
                    Array<T,3> iVertex = iTriangle[v];
                    if(iVertex[0] < xmin){ xmin = iVertex[0]; }
                    if(iVertex[0] > xmax){ xmax = iVertex[0]; }
                    if(iVertex[1] < ymin){ ymin = iVertex[1]; }
                    if(iVertex[1] > ymax){ ymax = iVertex[1]; }
                    if(iVertex[2] < zmin){ zmin = iVertex[2]; }
                    if(iVertex[2] > zmax){ zmax = iVertex[2]; }
                }
            }
            d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return d;
    }

}// namespace plb

#endif //WALL_H
```

variables.h 07/12/2016, 13:30

A.8. Variables

```
#ifndef VARIABLES_H
#define VARIABLES_H

#include <palabos3D.h>
#include "myheaders3D.h"

namespace plb{

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
class Variables{
private:
public:
    static int objCount;

    Variables();

    ~Variables();
// Methods
    static void initialize();

    void update(const plint& _gridLevel, const plint& _reynolds);

    bool checkConvergence();

    T getRho(const T& temp);

    std::unique_ptr<DEFscaledMesh<T> > createMesh(ConnectedTriangleSet<T>&
        triangleSet, const plint& referenceDirection,
        const int& flowType);

    std::unique_ptr<TriangleBoundary3D<T> > createTB(const DEFscaledMesh<T>&
        mesh);

    std::unique_ptr<VoxelizedDomain3D<T> > createVoxels(const
        TriangleBoundary3D<T>& tb, const int& flowType, const bool& dynamic)
        ;

    std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > createBP(const
        TriangleBoundary3D<T>& tb);

    std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > createFS(const
        VoxelizedDomain3D<T>& vozelizedDomain,
        const BoundaryProfiles3D<T,SurfaceData>* profile);

    std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > createModel
        (TriangleFlowShape3D<T,SurfaceData>* flowShape,
        const int& flowType);

    void createLattice(const VoxelizedDomain3D<T>& wallVoxels, const
        VoxelizedDomain3D<T>& obstacleVoxels);

    std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,BoundaryType>
        > createBC(
        GuoOffLatticeModel3D<T,Descriptor>* model,
        VoxelizedDomain3D<T>& vozelizedDomain);

    void join();

    void initializeLattice();
```

```
    void makeParallel();

    void setLattice();

    void save();

    void updateLattice();

    MultiContainerBlock3D* getContainer(){ return container;}

// Attributes
    static MultiContainerBlock3D* container;
    static T time, dx, dt, resolution, gridLevel, reynolds, scaled_u0lb;
    static plint iter;
    static Array<T,3> location;
    static plint nx, ny, nz;
    static Box3D boundingBox;
    static double scalingFactor;
    static std::vector<MultiBlock3D*> rhoBarJarg;
    static std::vector<MultiTensorField3D<T,3> > velocity, vorticity;
    static std::vector<MultiScalarField3D<T> > density;
    static IncomprFlowParam<T> p;
    static std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > lattice;
    static std::unique_ptr<MultiScalarField3D<T> > rhoBar;
    static std::unique_ptr<MultiTensorField3D<T,3> > j;
    static std::unique_ptr<IncBGKdynamics<T,Descriptor> > dynamics;
    static std::unique_ptr<Variables<T,BoundaryType,SurfaceData,Descriptor>
        > v;
private:
    static int nprocs, nprocs_side;
    static bool master;
};

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Variables<T,BoundaryType,SurfaceData,Descriptor>::objCount= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::time= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::resolution= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::gridLevel= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::reynolds= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::dx= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
```

```
T Variables<T,BoundaryType,SurfaceData,Descriptor>::dt= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::scaled_u0lb= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Variables<T,BoundaryType,SurfaceData,Descriptor>::iter= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Variables<T,BoundaryType,SurfaceData,Descriptor>::nx= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Variables<T,BoundaryType,SurfaceData,Descriptor>::ny= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Variables<T,BoundaryType,SurfaceData,Descriptor>::nz= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Array<T,3> Variables<T,BoundaryType,SurfaceData,Descriptor>::location= Array
    <T,3>();

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
Box3D Variables<T,BoundaryType,SurfaceData,Descriptor>::boundingBox= Box3D()
    ;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
double Variables<T,BoundaryType,SurfaceData,Descriptor>::scalingFactor= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Variables<T,BoundaryType,SurfaceData,Descriptor>::nprocs= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Variables<T,BoundaryType,SurfaceData,Descriptor>::nprocs_side= 0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<MultiBlock3D*> Variables<T,BoundaryType,SurfaceData,Descriptor>
    ::rhoBarJarg;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<MultiTensorField3D<T,3> > Variables<T,BoundaryType,SurfaceData,
    Descriptor>::velocity;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::vector<MultiTensorField3D<T,3> > Variables<T,BoundaryType,SurfaceData,
    Descriptor>::vorticity;

template<typename T, class BoundaryType, class SurfaceData, template<class U
```

```
    > class Descriptor>
std::vector<MultiScalarField3D<T> > Variables<T,BoundaryType,SurfaceData,
    Descriptor>::density;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Variables<T,BoundaryType,SurfaceData,Descriptor>::master= false;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
IncomprFlowParam<T> Variables<T,BoundaryType,SurfaceData,Descriptor>::p =
    IncomprFlowParam<T>(scaled_u0lb,reynolds,resolution,1,1,1);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiBlockLattice3D<T,Descriptor> > Variables<T,BoundaryType
    ,SurfaceData,Descriptor>::lattice(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiScalarField3D<T> >  Variables<T,BoundaryType,
    SurfaceData,Descriptor>::rhoBar(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiTensorField3D<T,3> > Variables<T,BoundaryType,
    SurfaceData,Descriptor>::j(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<IncBGKdynamics<T,Descriptor> > Variables<T,BoundaryType,
    SurfaceData,Descriptor>::dynamics(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
MultiContainerBlock3D* Variables<T,BoundaryType,SurfaceData,Descriptor>::
    container = nullptr;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<Variables<T,BoundaryType,SurfaceData,Descriptor> >
    Variables<T,BoundaryType,SurfaceData,Descriptor>::v(nullptr);

} // namespace plb

#endif // VARIABLES_H
```

variables.hh 07/12/2016, 16:08

```
#ifndef VARIABLES_HH
#define VARIABLES_HH

#include "variables.h"
#include <palabos3D.hh>
#include "myheaders3D.hh"

#include <thread>
#include <chrono>

namespace plb{

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Variables<T,BoundaryType,SurfaceData,Descriptor>::Variables()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Variables";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            this->v.reset(this);
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Variables already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Variables<T,BoundaryType,SurfaceData,Descriptor>::~Variables()
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Variables";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        objCount--;
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Variables<T,BoundaryType,SurfaceData,Descriptor>::initialize()
    {
        try{
            resolution = 0; gridLevel=0; reynolds=0;
            location = Array<T,3>();
            dx = 1;
            dt = 1;
            iter = 0;
```

```
            nprocs = 0;
            nprocs_side = 0;
            master = global::mpi().isMainProcessor();
            nprocs = global::mpi().getSize();
            nprocs_side = (int)cbrt(nprocs);
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Variables<T,BoundaryType,SurfaceData,Descriptor>::update(const
        plint& _gridLevel, const plint& _reynolds)
    {
        try{
            Constants<T>* c = Constants<T>::c.get();
            #ifdef PLB_DEBUG
                std::string mesg ="[DEBUG] Updating Parameters";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            gridLevel = _gridLevel;
            resolution = Constants<T>::physical.resolution * util::
                twoToThePowerPlint(_gridLevel);
            scaled_u0lb = Constants<T>::lb.u * util::twoToThePowerPlint
                (_gridLevel);
            reynolds = _reynolds;
            p = IncomprFlowParam<T>(Constants<T>::physical.u,scaled_u0lb,
                reynolds,Constants<T>::physical.length,
                Constants<T>::physical.resolution,Constants<T>::lb.lx,
                    Constants<T>::lb.ly,Constants<T>::lb.lz);
            dynamics.reset(new IncBGKdynamics<T,Descriptor>(p.getOmega()));
            dx = p.getDeltaX();
            dt = p.getDeltaT();
            T U = p.getLatticeU();
            scalingFactor = (T)(resolution)/dx;
            //for(int i = 0; i<rhoBarJarg.size(); i++){ delete
                rhoBarJarg[i];}
            rhoBarJarg.clear();
            velocity.clear();
            vorticity.clear();
            density.clear();
            //lattice.reset(nullptr);
            //rhoBar.reset(nullptr);
            //j.reset(nullptr);
            //container.reset(nullptr);
            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Reynolds="+std::to_string(reynolds);
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg = "[DEBUG] Grid Level="+std::to_string(gridLevel);
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg = "[DEBUG] Resolution="+std::to_string(resolution);
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg = "[DEBUG] Lattice U="+std::to_string(U);
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
```

```
            mesg = "[DEBUG] Lattice dt="+std::to_string(dt);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg = "[DEBUG] Lattice dx="+std::to_string(dx);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg = "[DEBUG] Scaling Factor="+std::to_string
                (scalingFactor);
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg = "[DEBUG] Done Updating Parameters";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        if(U == 0 || dx == 0 || dt == 0){
            writeLogFile(p, "WRONG PARAMATERS");
            throw std::runtime_error("InComprFlowParam not set
                Correctly");
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
bool Variables<T,BoundaryType,SurfaceData,Descriptor>::checkConvergence
    ()
{
    try{
        util::ValueTracer<T> tracer(p.getLatticeU(), p.getDeltaX(),
            Constants<T>::epsilon);
        return tracer.hasConverged();
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return false;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
T Variables<T,BoundaryType,SurfaceData,Descriptor>::getRho(const T& temp
    )
{
    T rho_lb = (T)1.0;
    try{
        const T dx = p.getDeltaX();
        T rho = 2579.3 - 0.6237*temp;
        rho_lb = rho*dx*dx*dx;
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return rho_lb;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<DEFscaledMesh<T> > Variables<T,BoundaryType,SurfaceData,
    Descriptor>::createMesh(
    ConnectedTriangleSet<T>& triangleSet, const plint&
```

```
        referenceDirection, const int& flowType)
{
    std::unique_ptr<DEFscaledMesh<T> > mesh(nullptr);
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Mesh";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").start();
        #endif
        // Create Mesh

        T dx = p.getDeltaX();
        T alpha = 1/dx;
        TriangleSet<T> normalTriangleSet = *triangleSet.toTriangleSet
            (Constants<T>::precision);

        Cuboid<T> cube = normalTriangleSet.getBoundingCuboid();
        Array<T,3> lowerLeftCorner = cube.lowerLeftCorner;
        Array<T,3> upperRightCorner = cube.upperRightCorner;

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Bounded Cuboid BEFORE Scaling Lower Left
                Corner "+array_string(lowerLeftCorner)+" Upper Right
                Corner "+
                array_string(upperRightCorner)+" in physical units";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        normalTriangleSet.scale(alpha);

        cube = normalTriangleSet.getBoundingCuboid();
        lowerLeftCorner = cube.lowerLeftCorner;
        upperRightCorner = cube.upperRightCorner;
        nx = upperRightCorner[0]+1;
        ny = upperRightCorner[1]+1;
        nz = upperRightCorner[2]+1;

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Bounded Cuboid AFTER Scaling Lower Left
                Corner "+array_string(lowerLeftCorner)+" Upper Right
                Corner "+
                array_string(upperRightCorner)+" in dimensionless units"
                    ;
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        mesh.reset(new DEFscaledMesh<T>(normalTriangleSet, resolution,
            referenceDirection,
                    Constants<T>::margin, Constants<T>::extraLayer));

        triangleSet = ConnectedTriangleSet<T>(normalTriangleSet);

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Mesh address= "+adr_string(mesh.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
```

```
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return mesh;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<TriangleBoundary3D<T> > Variables<T,BoundaryType,
    SurfaceData,Descriptor>::createTB(const DEFscaledMesh<T>& mesh)
{
    std::unique_ptr<TriangleBoundary3D<T> > triangleBoundary(nullptr);
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Triangle Boundary";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
        // Create Mesh
        triangleBoundary.reset(new TriangleBoundary3D<T>(mesh,true));
        triangleBoundary->getMesh().inflate();
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] TriangleBoundary address= "+adr_string
                (triangleBoundary.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return triangleBoundary;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<VoxelizedDomain3D<T> > Variables<T,BoundaryType,
    SurfaceData,Descriptor>::createVoxels(const TriangleBoundary3D<T>&
    tb,
    const int& flowType, const bool& dynamic)
{
    std::unique_ptr<VoxelizedDomain3D<T> > voxelizedDomain(nullptr);
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Voxelized Domain";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            std::cout << "[DEBUG] TB Address=" << &tb << " FlowType="<<
```

```
                flowType<<" ExtraLayer="<<Constants<T>::extraLayer <<
                " Borderwidth= "<<Constants<T>::borderWidth<<"
                    EnvelopeWidth= "<<Constants<T>::envelopeWidth<<"
                    Blocksize= "<<
                Constants<T>::blockSize<<" GridLevel= "<<gridLevel<<"
                    Dynamic Mesh= "<<dynamic<<std::endl;
            global::timer("boundary").restart();
        #endif
        voxelizedDomain.reset(
            new VoxelizedDomain3D<T>(
                tb,
                flowType,
                Constants<T>::extraLayer,
                Constants<T>::borderWidth,
                Constants<T>::envelopeWidth,
                Constants<T>::blockSize,
                gridLevel,
                dynamic));
        MultiScalarField3D<int> flagMatrix((MultiBlock3D&)
            voxelizedDomain->getVoxelMatrix());
        if(flowType == voxelFlag::inside){
            setToConstant(flagMatrix, voxelizedDomain->getVoxelMatrix(),
                    voxelFlag::outside, flagMatrix.getBoundingBox(), 1);
            setToConstant(flagMatrix, voxelizedDomain->getVoxelMatrix(),
                voxelFlag::outerBorder, flagMatrix.getBoundingBox(), 1);
        }
        else{
            setToConstant(flagMatrix, voxelizedDomain->getVoxelMatrix(),
                    voxelFlag::inside, flagMatrix.getBoundingBox(), 1);
            setToConstant(flagMatrix, voxelizedDomain->getVoxelMatrix(),
                voxelFlag::innerBorder, flagMatrix.getBoundingBox(), 1);
        }
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] VoxelizedDomain  address= "+adr_string
                (voxelizedDomain.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return voxelizedDomain;
}

/*
template<typename T, class BoundaryType, class SurfaceData,
    template<class U> class Descriptor>
std::unique_ptr<MultiBlockLattice3D<T,Descriptor> >
    Variables<T,BoundaryType,SurfaceData,Descriptor>::createLattice(
    VoxelizedDomain3D<T>& voxelizedDomain)
{
    std::unique_ptr<MultiBlockLattice3D<T,Descriptor> >
        partial_lattice(nullptr);
    try{
```

```
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Partial Lattice";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif

        Dynamics<T,Descriptor>* d = dynamics.get();

        partial_lattice.reset(new MultiBlockLattice3D<T,Descriptor>(nx,
            ny, nz, d));
        partial_lattice->toggleInternalStatistics(false);

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Partial Lattice address=
                "+adr_string(partial_lattice.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime(
                ));
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e)
        {exHandler(e,__FILE__,__FUNCTION__,__LINE__);}
    return partial_lattice;
}*/

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > Variables<T,
    BoundaryType,SurfaceData,Descriptor>::createBP(
    const TriangleBoundary3D<T>& tb)
{
    std::unique_ptr<BoundaryProfiles3D<T,SurfaceData> > profile(nullptr)
        ;
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Boundary Profile";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif

        profile.reset(new BoundaryProfiles3D<T,SurfaceData>());
        profile->setWallProfile(NoSlipProfile3D<T>().clone());
        plint numTriangles = tb.getMesh().getNumTriangles();
        int n=(int)numTriangles;
        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Defining Profiles for "+safe_string(n)+"
                surface triangles";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        for(plint t = 0; t<numTriangles; t++){
            plint tag = tb.getTag(t);
            profile->defineProfile(tag, NoSlipProfile3D<T>().clone());
```

```
        }

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Boundary Profile address= "+adr_string
                (profile.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return profile;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > Variables<T,
    BoundaryType,SurfaceData,Descriptor>::createFS(
    const VoxelizedDomain3D<T>& voxelizedDomain, const
        BoundaryProfiles3D<T,SurfaceData>* profile)
{
    std::unique_ptr<TriangleFlowShape3D<T,SurfaceData> > flowShape
        (nullptr);
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Triangle Flow Shape";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif

        if(profile == nullptr){ throw std::runtime_error("Boundary
            Profiles where destroyed!"); }

        flowShape.reset(
            new TriangleFlowShape3D<T,SurfaceData>(
                voxelizedDomain.getBoundary(),
                *profile)
        );

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Triangle Flow Shape address= "+adr_string
                (flowShape.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
```

```
        __LINE__);}
    return flowShape;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > Variables<T,
    BoundaryType,SurfaceData,Descriptor>::createModel(
    TriangleFlowShape3D<T,SurfaceData>* flowShape, const int& flowType)
{
    std::unique_ptr<GuoOffLatticeModel3D<T,Descriptor> > model(nullptr);
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Model";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif

        model.reset(
            new GuoOffLatticeModel3D<T,Descriptor>(
                flowShape,
                flowType)
        );

        model->setVelIsJ(true); // When the incompressible BGK model is
            used, velocity equals momentum.
        model->selectUseRegularizedModel(true);
        model->selectComputeStat(false);

        #ifdef PLB_DEBUG
            mesg ="[DEBUG] Model address= "+adr_string(model.get());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            mesg ="[DEBUG] Elapsed
                Time="+std::to_string(global::timer("boundary").getTime
                ());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("boundary").restart();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return model;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Variables<T,BoundaryType,SurfaceData,Descriptor>::createLattice
    (const VoxelizedDomain3D<T>& wallVoxels,
    const VoxelizedDomain3D<T>& obstacleVoxels)
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Joining Lattices ";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("join").start();
        #endif
```

```
MultiScalarField3D<int> wallMatrix = wallVoxels.getVoxelMatrix()
    ;
MultiScalarField3D<int> obstacleMatrix = obstacleVoxels.
    getVoxelMatrix();

Box3D fromDomain = Obstacle<T,BoundaryType,SurfaceData,
    Descriptor>::getDomain();
Box3D toDomain = Wall<T,BoundaryType,SurfaceData,Descriptor>::
    getDomain();

wallMatrix.copyReceive(obstacleMatrix,fromDomain,toDomain,
    modif::allVariables);

lattice.reset(generateMultiBlockLattice<T,Descriptor>(wallMatrix
    , Constants<T>::envelopeWidth, dynamics->clone()).release())
    ;

T resolution = Constants<T>::physical.resolution * util::
    twoToThePowerPlint(gridLevel);
T scaled_u0lb = Constants<T>::lb.u / util::twoToThePowerPlint
    (gridLevel);
p = IncomprFlowParam<T>(Constants<T>::physical.u, scaled_u0lb,
    reynolds, Constants<T>::physical.length,
                        resolution, lattice->getNx(), lattice->
                            getNy(), lattice->getNz());
std::string fileName = "parameters_Re="+std::to_string((int)
    reynolds)+"_GridLvL="+std::to_string((int)gridLevel);
writeLogFile(p, fileName);

defineDynamics(*lattice, lattice->getBoundingBox(), dynamics->
    clone());
lattice->toggleInternalStatistics(false);
defineDynamics(*lattice, wallMatrix, lattice->getBoundingBox(),
    new NoDynamics<T,Descriptor>, voxelFlag::outside);
defineDynamics(*lattice, obstacleMatrix, lattice->getBoundingBox
    (), new NoDynamics<T,Descriptor>, voxelFlag::inside);

rhoBar.reset(generateMultiScalarField<T>((MultiBlock3D&) *
    lattice, Constants<T>::envelopeWidth).release());
rhoBar->toggleInternalStatistics(false);

j.reset(generateMultiTensorField<T,3>((MultiBlock3D&) *lattice,
    Constants<T>::envelopeWidth).release());
j->toggleInternalStatistics(false);

rhoBarJarg.clear();
rhoBarJarg.push_back(dynamic_cast<MultiBlock3D*>(lattice.get()))
    ;
rhoBarJarg.push_back(dynamic_cast<MultiBlock3D*>(rhoBar.get()));
rhoBarJarg.push_back(dynamic_cast<MultiBlock3D*>(j.get()));

integrateProcessingFunctional(new ExternalRhoJcollideAndStream3D
    <T,Descriptor>(),lattice->getBoundingBox(), rhoBarJarg, 0);
integrateProcessingFunctional(new BoxRhoBarJfunctional3D<T,
    Descriptor>(), lattice->getBoundingBox(), rhoBarJarg, 3);

lattice->periodicity().toggleAll(false);
rhoBar->periodicity().toggleAll(false);
j->periodicity().toggleAll(false);
```

```
container = new MultiContainerBlock3D(*rhoBar);

T lx = lattice->getNx();
T ly = lattice->getNy();
T lz = lattice->getNz();

Box3D domain = lattice->getBoundingBox();

TriangularSurfaceMesh<T> mesh = Obstacle<T,BoundaryType,
    SurfaceData,Descriptor>::tb->getMesh();
//ConnectedTriangleSet<T> triangles =
    ConnectedTriangleSet<T>(mesh.toTriangleSet(Constants<T>::pre
    cision));

T numVertices = Obstacle<T,BoundaryType,SurfaceData,Descriptor>
    ::tb->getMesh().getNumVertices();
//T numVertices = triangles.getNumVertices();
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::numVertices =
    numVertices;

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vertices.clear
    ();
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vertices.resize
    (numVertices);
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vertices.
    reserve(numVertices);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::unitNormals.
    clear();
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::unitNormals.
    resize(numVertices);
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::unitNormals.
    reserve(numVertices);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::areas.clear();
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::areas.resize
    (numVertices);
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::areas.reserve
    (numVertices);

const bool weightedArea = false;

for(int i = 0; i < numVertices; i++){
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vertices[i]
        = mesh.getVertex(i);
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::areas[i] =
        mesh.computeVertexArea(i);
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::unitNormals
        [i] = mesh.computeVertexNormal(i,weightedArea);
    /*
    Array<T,3> v = Array<T,3>(0,0,0);
    v = triangles.getVertex(i);
    Array<T,3> n = Array<T,3>(0,0,0);
    T a = 0;
    triangles.computeVertexAreaAndUnitNormal(i, a, n);
    vertices[i] = v;
    areas[i] = a;
    unitNormals[i] = n;
    */
```

```
}

// Integrate the immersed boundary processors in the lattice
    multi-block.
std::vector<MultiBlock3D*> args;
plint pl = 4;

args.resize(0);
args.push_back(container);
integrateProcessingFunctional(new InstantiateImmersedWallData3D<
    T>(Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vertices
    ,
                                Obstacle<T,BoundaryType,
                                    SurfaceData,Descriptor>::
                                    areas,
                                Obstacle<T,BoundaryType,
                                    SurfaceData,Descriptor>::
                                    unitNormals),
                                container->getBoundingBox(), *
                                    lattice, args, pl);
//lattice->executeInternalProcessors(pl);
//instantiateImmersedWallData(vertices, areas, *container);

pl++;
// Update the Velocity Function once
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::velocityFunc.
    update(p,(T)0,Array<T,3>(0,0,0),Array<T,3>(0,0,0),
        Obstacle<T,BoundaryType,SurfaceData,Descriptor>::tb.get
            (), lattice->getBoundingBox());

for (plint i = 0; i < Constants<T>::ibIter; i++) {
    args.resize(0);
    args.push_back(rhoBar.get());
    args.push_back(j.get());
    args.push_back(container);
    integrateProcessingFunctional(
        new IndexedInamuroIteration3D<T,SurfaceVelocity<T> >(
            Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
                velocityFunc, p.getTau(), true),
            rhoBar->getBoundingBox(), *lattice, args, pl);
    pl++;
}
Box3D newDomain = lattice->getBoundingBox();

if(newDomain.x0 > fromDomain.x0 || newDomain.x1 < fromDomain.x1
|| newDomain.y0 > fromDomain.y0 || newDomain.y1 < fromDomain.y1
|| newDomain.z0 > fromDomain.z0 || newDomain.z1 < fromDomain.z1
|| newDomain.x0 > toDomain.x0 || newDomain.x1 < toDomain.x1
|| newDomain.y0 > toDomain.y0 || newDomain.y1 < toDomain.y1
|| newDomain.z0 > toDomain.z0 || newDomain.z1 < toDomain.z1)
{
    std::string ex = "[ERROR] Domain mismatch Lattice = "+
        box_string(newDomain)+" but obstacle = "
    +box_string(fromDomain)+" and wall = "+box_string(toDomain);
    throw std::runtime_error(ex);
}

#ifdef PLB_DEBUG
    mesg = "[DEBUG] Domain Information Lattice = "+ box_string
        (newDomain)+" Obstacle = "
```

```
                    +box_string(fromDomain)+" and Wall = "+box_string
                        (toDomain);
                if(master){ std::cout << mesg << std::endl; }
                global::log(mesg);
                mesg = "[DEBUG] Domain= "+ box_string(domain)+" Nx= "+std::
                    to_string(lx)+" Ny= "+std::to_string(ly)+" Nz= "+std::
                    to_string(lz);
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg = "[DEBUG] Done Joining Lattices
                    time="+std::to_string(global::timer("join").getTime());
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                global::timer("join").stop();
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,BoundaryType>
        > Variables<T,BoundaryType,SurfaceData,Descriptor>::createBC(
        GuoOffLatticeModel3D<T,Descriptor>* model, VoxelizedDomain3D<T>&
            voxelizedDomain)
    {
        std::unique_ptr<OffLatticeBoundaryCondition3D<T,Descriptor,
            BoundaryType> > boundaryCondition(nullptr);
        try{
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Creating BoundaryCondition";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                global::timer("boundary").restart();
            #endif

            boundaryCondition.reset(
                    new OffLatticeBoundaryCondition3D<T,Descriptor,
                        BoundaryType>(
                    model,
                    voxelizedDomain,
                    *lattice)
            );

            boundaryCondition->insert(rhoBarJarg);

            #ifdef PLB_DEBUG
                mesg ="[DEBUG] BoundaryCondition address= "+adr_string
                    (boundaryCondition.get());
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                mesg ="[DEBUG] Elapsed
                    Time="+std::to_string(global::timer("boundary").getTime
                    ());
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                global::timer("boundary").restart();
            #endif
        }
```

```
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return boundaryCondition;
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Variables<T,BoundaryType,SurfaceData,Descriptor>::initializeLattice
    ()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Initializing Lattice";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("ini").start();
        #endif

        T iniT = Constants<T>::initialTemperature;
        T rho_lb = getRho(iniT);
        Array<T,3> iniV = Array<T,3>(0,0,0);

        initializeAtThermalEquilibrium(*lattice, lattice->getBoundingBox
            (), rho_lb, iniV, iniT);
        //lattice->initialize();
        applyProcessingFunctional(new BoxRhoBarJfunctional3D<T,
            Descriptor>(),lattice->getBoundingBox(), rhoBarJarg);

        Wall<T,BoundaryType,SurfaceData,Descriptor>::bc->apply
            (rhoBarJarg);
        Obstacle<T,BoundaryType,SurfaceData,Descriptor>::bc->apply
            (rhoBarJarg);

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Initializing Lattice time="+std::
                to_string(global::timer("join").getTime());
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("ini").stop();
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Variables<T,BoundaryType,SurfaceData,Descriptor>::makeParallel()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Parallelizing Lattice ";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            global::timer("parallel").start();
        #endif
        Box3D box = lattice->getBoundingBox();
        std::map<plint, BlockLattice3D< T, Descriptor > * > blockMap =
            lattice->getBlockLattices();
        plint size = blockMap.size();
```

```
            std::vector< std::vector<Box3D> > domains;
            domains.resize(size);

            for(int i=0; i<size; i++){
                std::vector<Box3D> block;
                plint nx = blockMap[i]->getNx()-1;
                plint ny = blockMap[i]->getNy()-1;
                plint nz = blockMap[i]->getNz()-1;
                for(int x = 0; x<nx; x++){
                    for(int y = 0; y<nx; y++){
                        for(int z=0; z<nz; z++){
                            Box3D cell(x,x+1,y,y+1,z,z+1);
                            block.push_back(cell);
                        }
                    }
                }
                domains.push_back(block);
            }
            ParallellizeByCubes3D parallel(domains, box, nprocs_side,
                nprocs_side, nprocs_side);
            parallel.parallelize();
            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Done Parallelizing Lattice time="+std::
                    to_string(global::timer("parallel").getTime());
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
                global::timer("parallel").stop();
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Variables<T,BoundaryType,SurfaceData,Descriptor>::setLattice()
    {
        try{
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Main Lattice";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif

            std::unique_ptr<DEFscaledMesh<T> > wall_mesh = createMesh(Wall<T
                ,BoundaryType,SurfaceData,Descriptor>::triangleSet,
                Constants<T>::wall.referenceDirection, Wall<T,BoundaryType,
                    SurfaceData,Descriptor>::flowType);

            std::unique_ptr<DEFscaledMesh<T> > obstacle_mesh = createMesh
                (Obstacle<T,BoundaryType,SurfaceData,Descriptor>::
                triangleSet,
                Constants<T>::obstacle.referenceDirection, Obstacle<T,
                    BoundaryType,SurfaceData,Descriptor>::flowType);

            Wall<T,BoundaryType,SurfaceData,Descriptor>::tb = createTB(*
                wall_mesh);
            Wall<T,BoundaryType,SurfaceData,Descriptor>::location = Wall<T,
                BoundaryType,SurfaceData,Descriptor>::tb->
                getPhysicalLocation();
```

```
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::tb = createTB(*
    obstacle_mesh);
Obstacle<T,BoundaryType,SurfaceData,Descriptor>::location =
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::tb->
    getPhysicalLocation();

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::moveToStart();

Wall<T,BoundaryType,SurfaceData,Descriptor>::vd = createVoxels(*
    Wall<T,BoundaryType,SurfaceData,Descriptor>::tb,
    Wall<T,BoundaryType,SurfaceData,Descriptor>::flowType,
        Constants<T>::wall.dynamicMesh);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vd =
    createVoxels(*Obstacle<T,BoundaryType,SurfaceData,Descriptor
    >::tb,
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::flowType,
        Constants<T>::obstacle.dynamicMesh);

createLattice(*Wall<T,BoundaryType,SurfaceData,Descriptor>::vd,
    *Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vd);

Wall<T,BoundaryType,SurfaceData,Descriptor>::bp = createBP(*Wall
    <T,BoundaryType,SurfaceData,Descriptor>::tb);

Wall<T,BoundaryType,SurfaceData,Descriptor>::fs = createFS(*Wall
    <T,BoundaryType,SurfaceData,Descriptor>::vd,
    Wall<T,BoundaryType,SurfaceData,Descriptor>::bp.get());

Wall<T,BoundaryType,SurfaceData,Descriptor>::model = createModel
    (Wall<T,BoundaryType,SurfaceData,Descriptor>::fs.get(),
    Wall<T,BoundaryType,SurfaceData,Descriptor>::flowType);

Wall<T,BoundaryType,SurfaceData,Descriptor>::bc = createBC(Wall<
    T,BoundaryType,SurfaceData,Descriptor>::model.get(),
    *Wall<T,BoundaryType,SurfaceData,Descriptor>::vd);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::bp = createBP(*
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::tb);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::fs = createFS(*
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vd,
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::bp.get());

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::model =
    createModel(Obstacle<T,BoundaryType,SurfaceData,Descriptor>
    ::fs.get(),
    Obstacle<T,BoundaryType,SurfaceData,Descriptor>::flowType);

Obstacle<T,BoundaryType,SurfaceData,Descriptor>::bc = createBC
    (Obstacle<T,BoundaryType,SurfaceData,Descriptor>::model.get
    (),
    *Obstacle<T,BoundaryType,SurfaceData,Descriptor>::vd);

initializeLattice();

//makeParallel();

#ifdef PLB_DEBUG
```

```
            mesg = "[DEBUG] Done Constructing Main Lattice";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Variables<T,BoundaryType,SurfaceData,Descriptor>::save()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Saving Data";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        lattice->toggleInternalStatistics(false);

        //MultiTensorField3D<T, 3> v = *computeVelocity(*lattice);
        //velocity.push_back(v);

        //MultiTensorField3D<T, 3> w = *computeVorticity(v);
        //vorticity.push_back(w);

        //MultiScalarField3D<T> r = *computeDensity(*lattice);
        //density.push_back(r);

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Saving Data";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Variables<T,BoundaryType,SurfaceData,Descriptor>::updateLattice()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Updating Main Lattice";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        lattice->toggleInternalStatistics(false);

        //save();

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Updating Main Lattice";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
```

```
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }


} // namespace plb

#endif // VARIABLES_HH
```

output.h 07/12/2016, 13:29

A.9. Output

```
#ifndef OUTPUT_H
#define OUTPUT_H

#include <palabos3D.h>
#include "myheaders3D.h"

namespace plb{

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
class Output{
private:
public:
    static int objCount;

    explicit Output();

    ~Output();
    // Methods
    static void initialize();

    void elapsedTime();

    void timeLoop();

    void writeGif();
private:
    void writeDensity();

    void writeVelocity();

    void writeVorticity();

public:
    void writeImages(const plint& reynolds_, const plint& gridLevel_, const
        bool& last = false);

    void startMessage();

    void simMessage();

    void stopMessage();

    static std::unique_ptr<Output<T,BoundaryType,SurfaceData,Descriptor> >
        out;
private:
    static std::unique_ptr<VtkStructuredImageOutput3D<T> > densityOut;
    static std::unique_ptr<VtkStructuredImageOutput3D<T> > velocityOut;
    static std::unique_ptr<VtkStructuredImageOutput3D<T> > vorticityOut;
    static std::unique_ptr<MultiTensorField3D<T,3> > v;
    static std::unique_ptr<MultiTensorField3D<T,3> > w;
    static std::unique_ptr<MultiScalarField3D<T> > r;
    static plint reynolds;
    static plint gridLevel;
    static bool master;
    static bool first;
    static bool last;
    global::PlbTimer timer;
    static double startTime;
    static double endTime;
```

```
    static int gifCount;
    static int vtkCount;
};

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Output<T,BoundaryType,SurfaceData,Descriptor>::objCount=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<VtkStructuredImageOutput3D<T> > Output<T,BoundaryType,
    SurfaceData,Descriptor>::densityOut(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<VtkStructuredImageOutput3D<T> > Output<T,BoundaryType,
    SurfaceData,Descriptor>::velocityOut(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<VtkStructuredImageOutput3D<T> > Output<T,BoundaryType,
    SurfaceData,Descriptor>::vorticityOut(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiTensorField3D<T,3> > Output<T,BoundaryType,SurfaceData,
    Descriptor>::v(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiTensorField3D<T,3> > Output<T,BoundaryType,SurfaceData,
    Descriptor>::w(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<MultiScalarField3D<T> > Output<T,BoundaryType,SurfaceData,
    Descriptor>::r(nullptr);

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Output<T,BoundaryType,SurfaceData,Descriptor>::reynolds=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
plint Output<T,BoundaryType,SurfaceData,Descriptor>::gridLevel=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Output<T,BoundaryType,SurfaceData,Descriptor>::gifCount=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
int Output<T,BoundaryType,SurfaceData,Descriptor>::vtkCount=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Output<T,BoundaryType,SurfaceData,Descriptor>::master=false;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
```

```
bool Output<T,BoundaryType,SurfaceData,Descriptor>::first=true;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
bool Output<T,BoundaryType,SurfaceData,Descriptor>::last=false;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
double Output<T,BoundaryType,SurfaceData,Descriptor>::startTime=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
double Output<T,BoundaryType,SurfaceData,Descriptor>::endTime=0;

template<typename T, class BoundaryType, class SurfaceData, template<class U
    > class Descriptor>
std::unique_ptr<Output<T,BoundaryType,SurfaceData,Descriptor> > Output<T,
    BoundaryType,SurfaceData,Descriptor>::out(nullptr);

} // namespace plb

#endif // OUTPUT_H
```

output.hh 07/12/2016, 13:29

```
#ifndef OUTPUT_HH
#define OUTPUT_HH

#include "output.h"
#include <palabos3D.hh>
#include "myheaders3D.hh"

#include <thread>
#include <ctime>
#include <chrono>
#include <string>
#include <exception>
#include <iostream>
#include <iomanip>

namespace plb{

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Output<T,BoundaryType,SurfaceData,Descriptor>::Output()
    {
        if(objCount==0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing Output";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            this->out.reset(this);
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Output already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    Output<T,BoundaryType,SurfaceData,Descriptor>::~Output()
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Destroying Output";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        objCount--;
    }


    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Output<T,BoundaryType,SurfaceData,Descriptor>::initialize()
    {
```

```
    try{
        bool parallel = false;
        #ifdef PLB_MPI_PARALLEL
            parallel = true;
        #endif
        global::log().init(parallel);
        master = global::mpi().isMainProcessor();
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::elapsedTime()
{
    try{
        if(Constants<T>::test){std::thread(timeLoop);}
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::timeLoop()
{
    try{
        double elapsed = 0;
        double testTime = Constants<T>::testTime*60;
        bool running = true;
        while(running){
            elapsed = timer.getTime() - startTime;
            if(elapsed > testTime){ throw std::runtime_error("Test Time
                Limit Exceeded!"); running = false; }
            std::this_thread::sleep_for(std::chrono::minutes(1));
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::writeGif()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Writing GIF";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        const plint imSize = 600;
        plint nx = Variables<T,BoundaryType,SurfaceData,Descriptor>::
            lattice->getNx();
        plint ny = Variables<T,BoundaryType,SurfaceData,Descriptor>::
            lattice->getNy();
        plint nz = Variables<T,BoundaryType,SurfaceData,Descriptor>::
            lattice->getNz();
        nx = nx/2;
```

```
        ny = ny/2;
        nz = nz/2;
        Box3D slice(nx, nx-1, ny, ny-1, nz, nz);
        ImageWriter<T> imageWriter("leeloo");
        imageWriter.writeScaledGif(createFileName("u", gifCount, 6),
            *computeVelocityNorm(*Variables<T,BoundaryType,SurfaceData,
                Descriptor>::lattice, slice),imSize, imSize );
        gifCount++;
        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Writing GIF";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}


template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::writeDensity()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Writing Density";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaX();
        const T dt = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaT();

        float tconv =  (float)dx/dt;
        float offset = (float)0;

        r.reset(computeDensity(*Variables<T,BoundaryType,SurfaceData,
            Descriptor>::lattice).release());
        std::string name = "density";
        densityOut->writeData(*r, name, tconv, offset, first, last);

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Writing Density";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::writeVelocity()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Writing Velocity";
            if(master){std::cout << mesg << std::endl;}
```

```
            global::log(mesg);
        #endif
        const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaX();
        const T dt = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaT();

        float tconv =  (float)dx/dt;
        float offset = (float)0;

        //Tensor field for the velocity
        std::string name = "velocity";
        v.reset(computeVelocity(*Variables<T,BoundaryType,SurfaceData,
            Descriptor>::lattice).release());
        velocityOut->writeData(*v, name, tconv, first, last);

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Writing Velocity";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
void Output<T,BoundaryType,SurfaceData,Descriptor>::writeVorticity()
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Writing Vorticity";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaX();
        const T dt = Variables<T,BoundaryType,SurfaceData,Descriptor>::p
            .getDeltaT();

        float tconv =  (float)dx/dt;
        float offset = (float)0;

        w.reset(computeVorticity(*v).release());
        std::string name = "vorticity";
        vorticityOut->writeData(*w, name, tconv, first, last);

        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Done Writing Vorticity";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<typename T, class BoundaryType, class SurfaceData, template<
    class U> class Descriptor>
```

output.hh 07/12/2016, 13:29

```
void Output<T,BoundaryType,SurfaceData,Descriptor>::writeImages(const
    plint& reynolds_, const plint& gridLevel_, const bool& last_)
{
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Writing VTK";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        last = last_;
        if(vtkCount==0)
        {
            first = true;
            reynolds = reynolds_;
            gridLevel = gridLevel_;
            const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor
                >::p.getDeltaX();

            std::string fileName = "density_Re"+std::to_string(reynolds)
                +"_Lvl"+std::to_string(gridLevel)+".dat";
            densityOut.reset(new VtkStructuredImageOutput3D<T>(fileName,
                dx));
            pcout << "1" << std::endl;
            fileName = "velocity_Re"+std::to_string(reynolds)+"_Lvl"+
                std::to_string(gridLevel)+".dat";
            velocityOut.reset(new VtkStructuredImageOutput3D<T>(fileName
                , dx));
            pcout << "2" << std::endl;
            fileName = "vorticity_Re"+std::to_string(reynolds)+"_Lvl"+
                std::to_string(gridLevel)+".dat";
            vorticityOut.reset(new VtkStructuredImageOutput3D<T>
                (fileName, dx));
        }
        else if(reynolds != reynolds_ || gridLevel != gridLevel_){
            first = true;
            reynolds = reynolds_;
            gridLevel = gridLevel_;
            const T dx = Variables<T,BoundaryType,SurfaceData,Descriptor
                >::p.getDeltaX();

            std::string fileName = "density_Re"+std::to_string(reynolds)
                +"_Lvl"+std::to_string(gridLevel)+".dat";
            densityOut.reset(new VtkStructuredImageOutput3D<T>(fileName,
                dx));

            fileName = "velocity_Re"+std::to_string(reynolds)+"_Lvl"+
                std::to_string(gridLevel)+".dat";
            velocityOut.reset(new VtkStructuredImageOutput3D<T>(fileName
                , dx));

            fileName = "vorticity_Re"+std::to_string(reynolds)+"_Lvl"+
                std::to_string(gridLevel)+".dat";
            vorticityOut.reset(new VtkStructuredImageOutput3D<T>
                (fileName, dx));
            vtkCount = 0;
        }
        else{first = false;}

        writeDensity();
```

```
            writeVelocity();

            writeVorticity();

            vtkCount++;

            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Done Writing VTK";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Output<T,BoundaryType,SurfaceData,Descriptor>::startMessage()
    {
        try
        {
            time_t rawtime;
            struct tm * timeinfo;
            time (&rawtime);
            timeinfo = localtime (&rawtime);
            if(master)
            {
                std::cout<<"SIMULATION START "<< asctime(timeinfo) << std::
                    endl;
            }
            #ifdef PLB_MPI_PARALLEL
            if(master){
                int size = plb::global::mpi().getSize();
                std::cout<<"NUMBER OF MPI PROCESSORS="<< size << std::endl;
                //if((int)cbrt(size) % 1){ throw std::runtime_error("Number
                    of MPI Processess must satisfy Cubic Root");}
                std::string imaster =  master ? " YES " : " NO ";
                std::cout<<"Is this the main process?"<< imaster << std::
                    endl;
            }
            #endif
            std::string outputDir = "./tmp/";
            global::directories().setOutputDir(outputDir);// Set output DIR
                w.r.t. to current DIR
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }


    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Output<T,BoundaryType,SurfaceData,Descriptor>::simMessage()
    {
        try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Creating Timer";
            if(master){std::cout << mesg << std::endl;}
```

```
            global::log(mesg);
        #endif
        timer.start();      // Start Timer
        startTime = timer.getTime();
        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Timer Started";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            if(Constants<T>::test){mesg = "[DEBUG] Starting Test";}
            else{mesg = "[DEBUG] Starting Normal Run";}
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    template<typename T, class BoundaryType, class SurfaceData, template<
        class U> class Descriptor>
    void Output<T,BoundaryType,SurfaceData,Descriptor>::stopMessage()
    {
        try{
        std::string mesg = "SIMULATION COMPLETE";
        if(master){std::cout << mesg << std::endl;}
        global::log(mesg);
        elapsedTime();
        timer.stop();
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

} // namespace plb

#endif // OUTPUT_HH
```

velocity.h 07/12/2016, 13:30

A.10. Velocity

```
#ifndef VELOCITY_H
#define VELOCITY_H

#include <palabos3D.h>
#include "myheaders3D.h"

namespace plb{

template<typename T>
struct Kinematics{
    Array<T,3> alpha_lb;
    Array<T,3> omega_lb;
    Array<T,3> a_lb;
    Array<T,3> v_lb;
};

template<typename T>
class SurfaceVelocity{
public:
    static int objCount;

    SurfaceVelocity();

    Array<T,3> operator()(pluint id);

    void initialize(const T& mass, const T& g, const T& rho);

    Box3D getDomain(const TriangleBoundary3D<T>* tb);

    Array<T,3> getCG(std::vector<Array<T,3> > vertexList);

    Array<T,3> getArm(const Array<T,3>& p1, const Array<T,3>& p2);

    Array<T,6> getMomentOfInertia(const Array<T,3>& cg, const
        TriangleBoundary3D<T>* tb, const T& rho_lb);

    Array<T,3> getAlpha(const Array<T,3>& M, const Array<T,6>& I);

    Array<T,3> getRotation(const Array<T,3>& vertex, const Array<T,3>& cg,
        const Array<T,3>& dtheta);

    Array<T,3> getTotalVelocity(const Array<T,3>& vertex, const Array<T,3>&
        cg, const Array<T,3>& omega_lb, const Array<T,3>& v_lb);

    bool outOfBounds(const Box3D& domain, const Array<T,3> vertex);

    bool update(const IncomprFlowParam<T>& p, const T& timeLB, const Array<T
        ,3>& force, const Array<T,3>& torque,
                        TriangleBoundary3D<T>* tb, const Box3D& domain);
// Attributes
private:
    static bool master;
    static bool rotation;
    static plint moves;
    static Kinematics<T> previous;
    static std::vector<Array<T,3> > verticesVelocity;
    static std::vector<Array<T,3> > forceList;
    static std::vector<Array<T,3> > torque;
    static std::vector<Array<T,3> > location;
    static std::vector<Array<T,3> > acceleration;
```

```
    static std::vector<Array<T,3> > velocity;
    static std::vector<Array<T,3> > angular_acceleration;
    static std::vector<Array<T,3> > angular_velocity;
    static std::vector<T> time;
    static T rho, mass, g;
};

// Initializers
template<typename T>
int SurfaceVelocity<T>::objCount= 0;

template<typename T>
bool SurfaceVelocity<T>::master= false;

template<typename T>
plint SurfaceVelocity<T>::moves= 0;

template<typename T>
Kinematics<T> SurfaceVelocity<T>::previous;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::verticesVelocity;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::forceList;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::torque;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::location;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::acceleration;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::velocity;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::angular_acceleration;

template<typename T>
std::vector<Array<T,3> > SurfaceVelocity<T>::angular_velocity;

template<typename T>
std::vector<T> SurfaceVelocity<T>::time;

template<typename T>
T SurfaceVelocity<T>::rho = (T)0;

template<typename T>
T SurfaceVelocity<T>::mass = (T)0;

template<typename T>
T SurfaceVelocity<T>::g = (T)9.81;

}// namespace plb


#endif //VELOCITY_H
```

velocity.hh 07/12/2016, 13:30

```
#ifndef VELOCITY_HH
#define VELOCITY_HH

#include <cmath>

namespace plb{

    template<typename T>
    SurfaceVelocity<T>::SurfaceVelocity()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing SurfaceVelocity";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Surface Velocity already defined"
                ;
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T>
    Array<T,3> SurfaceVelocity<T>::operator()(pluint id)
    {
        return verticesVelocity[id];
    }

    template<typename T>
    void SurfaceVelocity<T>::initialize(const T& mass_, const T& g_, const T
        & rho_)
    {
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Initializing SurfaceVelocity";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif

        mass = mass_;
        rho = rho_;
        g = -g;

        previous.a_lb = Array<T,3>(0,0,0);
        previous.v_lb = Array<T,3>(0,0,0);
        previous.alpha_lb = Array<T,3>(0,0,0);
        previous.omega_lb = Array<T,3>(0,0,0);

        Array<T,3> a = Array<T,3>(0, 0, g);
        acceleration.push_back(a);
        Array<T,3> f = Array<T,3>(0,0,g*mass);
```

```
    forceList.push_back(f);
    Array<T,3> m = Array<T,3>(0,0,0);
    torque.push_back(m);
    T t = 0;
    time.push_back(t);

    #ifdef PLB_DEBUG
        mesg = "[DEBUG] DONE Initializing SurfaceVelocity";
        if(master){std::cout << mesg << std::endl;}
        global::log(mesg);
    #endif
}


template<typename T>
Box3D SurfaceVelocity<T>::getDomain(const TriangleBoundary3D<T>* tb)
{
    Box3D d(0,0,0,0,0,0);
    try
    {
        T numVertices = tb->getMesh().getNumVertices();
        T zmin = std::numeric_limits<T>::max();
        T zmax = std::numeric_limits<T>::min();
        T ymin = std::numeric_limits<T>::max();
        T ymax = std::numeric_limits<T>::min();
        T xmin = std::numeric_limits<T>::max();
        T xmax = std::numeric_limits<T>::min();
        for(int i = 0; i<numVertices; i++){
            Array<T,3> iVertex = tb->getMesh().getVertex(i);
            if(iVertex[0] < xmin){ xmin = iVertex[0]; }
            if(iVertex[0] > xmax){ xmax = iVertex[0]; }
            if(iVertex[1] < ymin){ ymin = iVertex[1]; }
            if(iVertex[1] > ymax){ ymax = iVertex[1]; }
            if(iVertex[2] < zmin){ zmin = iVertex[2]; }
            if(iVertex[2] > zmax){ zmax = iVertex[2]; }
        }
        d = Box3D(xmin,xmax,ymin,ymax,zmin,zmax);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return d;
}

template<typename T>
Array<T,3> SurfaceVelocity<T>::getCG(std::vector<Array<T,3> > vertexList
    )
{
    // Compute the center of gravity
    Array<T,3> cg = Array<T,3>(0,0,0);
    try{
        T x = 0;
        T y = 0;
        T z = 0;
        int size = vertexList.size();

        for(int i = 0; i<size; i++){
            Array<T,3> iVertex = vertexList[i];
            x += iVertex[0];
            y += iVertex[1];
            z += iVertex[2];
```

```
        }

        cg = Array<T,3>(x/size, y/size, z/size);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return cg;
}

template<typename T>
Array<T,3> SurfaceVelocity<T>::getArm(const Array<T,3>& p1, const Array<
    T,3>& p2)
{
    // Compute the arm r between the position and cg
    Array<T,3> arm = Array<T,3>(0,0,0);
    try{
        for(int i = 0; i<3; i++){
            if(p1[i] > p2[i]){
                if(p1[i] >= 0){
                    arm[i] = p1[i] - p2[i];
                }
                if(p1[i] < 0 && p2[i] < 0){
                    T a = -1*p2[i];
                    T b = -1*p1[i];
                    arm[i] = a - b;
                }
            }
            else{
                if(p2[i] >= 0){
                    arm[i] = p2[i] - p1[i];
                }
                if(p1[i] < 0 && p2[i] < 0){
                    T a = -1*p1[i];
                    T b = -1*p2[i];
                    arm[i] = a - b;
                }
            }
            if(arm[i] < 0){ arm[i]*= -1; }
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return arm;
}

template<typename T>
Array<T,6> SurfaceVelocity<T>::getMomentOfInertia(const Array<T,3>& cg,
    const TriangleBoundary3D<T>* tb,
    const T& rho_lb )
{
    // Compute the moment of Inertia
    Array<T,6> I = Array<T,6>(0,0,0,0,0,0);
    try{
        T Ixx = 0;
        T Iyy = 0;
        T Izz = 0;
        T Ixy = 0;
        T Ixz = 0;
        T Iyz = 0;
        T numTriangles = tb->getMesh().getNumTriangles();
```

```
            for(int i = 0; i<numTriangles; i++){
                Array<T,3> a =  tb->getMesh().getVertex(i,0);
                Array<T,3> b =  tb->getMesh().getVertex(i,1);
                Array<T,3> c =  tb->getMesh().getVertex(i,2);
                Array<T,3> d = cg;
                T iVolume = computeTetrahedronSignedVolume(a,b,c,d);
                if(iVolume<0){iVolume *= -1; }
                T iMass = iVolume * rho_lb;
                Array<T,3> iCG = Array<T,3>(0,0,0);
                for(int i = 0; i<3; i++){
                    T temp = a[i]+b[i]+c[i]+d[i];
                    iCG[i] = temp/4;
                }
                T x = iCG[0] - cg[0];
                T y = iCG[1] - cg[1];
                T z = iCG[2] - cg[2];
                T temp = y*y + z*z;
                Ixx += iMass * temp;
                temp = x*x + z*z;
                Iyy += iMass * temp;
                temp = x*x + y*y;
                Izz += iMass * temp;
                Ixy += iMass * x * y;
                Ixz += iMass * x * z;
                Iyz += iMass * y * z;
            }
            Ixy *= -1;
            Ixz *= -1;
            Iyz *= -1;
            I[0] = Ixx;
            I[1] = Iyy;
            I[2] = Izz;
            I[3] = Ixy;
            I[4] = Ixz;
            I[5] = Iyz;
            #ifdef PLB_DEBUG
                pcout << "[DEBUG]: Moment of Inertia =" <<std::endl;
                pcout << "| "<< Ixx <<" "<<Ixy<<" "<<Ixz<<" |"<<std::endl;
                pcout << "| "<< Ixy <<" "<<Iyy<<" "<<Ixy<<" |"<<std::endl;
                pcout << "| "<< Ixz <<" "<<Ixy<<" "<<Izz<<" |"<<std::endl;
            #endif
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return I;
    }

    template<typename T>
    Array<T,3> SurfaceVelocity<T>::getAlpha(const Array<T,3>& M, const Array
        <T,6>& I)
    {
        // Compute the angular acceleration
        Array<T,3> alpha = Array<T,3>(0,0,0);
        try{
            if(M[0] != 0 || M[1] != 0 || M[2] != 0){
                T Ixx = I[0];
                T Iyy = I[1];
                T Izz = I[2];
                T Ixy = I[3]; // Ixy == Iyx
                T Ixz = I[4]; // Ixz == Izx
```

```
            T Iyz = I[5]; // Iyz == Izy
            T Tx = M[0];
            T Ty = M[1];
            T Tz = M[2];
            alpha[0] = Tx / Ixx + Ty / Ixy + Tz / Ixz;
            alpha[1] = Tx / Ixy + Ty / Iyy + Tz / Iyz;
            alpha[2] = Tx / Ixz + Ty / Iyz + Tz / Izz;
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return alpha;
}

template<typename T>
Array<T,3> SurfaceVelocity<T>::getRotation(const Array<T,3>& vertex,
    const Array<T,3>& cg, const Array<T,3>& dtheta)
{
    Array<T,3> newVertex = Array<T,3>(0,0,0);
    try{
        if(dtheta[0] != 0 || dtheta[1] != 0 || dtheta[2] != 0){
            // X, Y, Z Rotation
            const T PI = std::acos(-1);
            // 1. Translate c.g. to origin
            newVertex = vertex - cg;
            // 2. Rotation about the X-axis
            T dx = newVertex[0];
            T dy = newVertex[1]*std::cos(dtheta[0]) - newVertex[2]*std::
                sin(dtheta[0]);
            T dz = newVertex[1]*std::sin(dtheta[0]) + newVertex[2]*std::
                cos(dtheta[0]);
            newVertex = Array<T,3>(dx,dy,dz);
            // 3. Rotation about the Y-axis
            dx = newVertex[2]*std::sin(dtheta[1]) + newVertex[0]*std::
                cos(dtheta[1]);
            dy = newVertex[1];
            dz = newVertex[2]*std::cos(dtheta[1]) - newVertex[0]*std::
                sin(dtheta[1]);
            newVertex = Array<T,3>(dx,dy,dz);
            // 4. Rotation about the Z-axis
            dx = newVertex[0]*std::cos(dtheta[2]) - newVertex[1]*std::
                sin(dtheta[2]);
            dy = newVertex[0]*std::sin(dtheta[2]) + newVertex[1]*std::
                cos(dtheta[2]);
            dz = newVertex[2];
            newVertex = Array<T,3>(dx,dy,dz);
            // 5. Translate c.g. back to position
            newVertex += cg;
        }
        else{return vertex; }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return newVertex;
}

template<typename T>
Array<T,3> SurfaceVelocity<T>::getTotalVelocity(const Array<T,3>& vertex
    , const Array<T,3>& cg, const Array<T,3>& omega_lb,
    const Array<T,3>& v_lb)
```

```
{
    Array<T,3> v = Array<T,3>(0,0,0);
    try{
        if(omega_lb[0] != 0 || omega_lb[1] != 0 || omega_lb[2] !=0){
            // X, Y, Z Rotational Velocity to Linear
            T base = std::pow(vertex[0],2) - 2*cg[0]*vertex[0] + std::
                pow(cg[0],2);
            T dx = std::pow(base,0.5);
            base = std::pow(vertex[1],2) - 2*cg[1]*vertex[1] + std::pow
                (cg[1],2);
            T dy= std::pow(base,0.5);
            base = std::pow(vertex[2],2) - 2*cg[2]*vertex[2] + std::pow
                (cg[2],2);
            T dz = std::pow(base,0.5);
            // Cross Product of r X omega
            v[0] = omega_lb[1]*dz - omega_lb[2]*dy;
            v[1] = omega_lb[2]*dx - omega_lb[0]*dz;
            v[2] = omega_lb[0]*dz - omega_lb[2]*dx;
            v += v_lb;
        }
        else{ return v_lb; }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return v;
}

template<typename T>
bool SurfaceVelocity<T>::outOfBounds(const Box3D& domain, const Array<T,
    3> vertex)
{
    try{
            //if(vertex[0] < domain.x0 || vertex[0] > domain.x1){ return
                true;}
            //if(vertex[1] < domain.y0 || vertex[1] > domain.y1){ return
                true;}
            if(vertex[2] < domain.z0 || vertex[2] > domain.z1){ return
                true;}
        }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return false;
}

template<typename T>
bool SurfaceVelocity<T>::update(const IncomprFlowParam<T>& p, const T&
    time_lb, const Array<T,3>& force,
    const Array<T,3>& torque, TriangleBoundary3D<T>* tb, const Box3D&
        domain)
{
    bool stop = false;
    try{
        #ifdef PLB_DEBUG
            std::string mesg = "[DEBUG] Updating SurfaceVelocity";
            if(master){std::cout << mesg << std::endl;}
            global::log(mesg);
            pcout << "[DEBUG] Input in Dimensionless Units" << std::endl
                ;
            pcout << "[DEBUG] FluidForce= "<< array_string(force) <<
                std::endl;
```

```
    pcout << "[DEBUG] FluidTorque= "<< array_string(torque) <<
        std::endl;
    pcout << "[DEBUG] Lattice Domain= "<< box_string(domain) <<
        std::endl;
    pcout << "[DEBUG] Obstacle Domain= " << box_string(getDomain
        (tb)) << std::endl;
#endif

const T dt = p.getDeltaT();
const T dx = p.getDeltaX();

plint n = tb->getMesh().getNumVertices();
std::vector<Array<T,3> > oldVertices;
oldVertices.resize(n);
oldVertices.reserve(n);

for(plint i = 0; i<n; i++){oldVertices[i] = tb->getMesh().
    getVertex(i);}

Array<T,3> cg_lb = getCG(oldVertices);


T dx3 = dx*dx*dx;
T dt2 = dt * dt;
T g_conv = dt2 / dx;

T rho_lb = rho / dx3;
T mass_lb = mass / dx3;
T g_lb = g * g_conv;
T gravityForce = mass_lb * g_lb;

Array<T,3> f_lb = Array<T,3>(0,0,0);
f_lb = force; // * dt*dt / (dx * dx * dx *dx);
f_lb[2] += gravityForce;

Array<T,3> torque_lb = Array<T,3>(0,0,0);
torque_lb = torque; // * dt*dt / (dx * dx * dx * dx * dx);

Array<T,6> I_lb = Array<T,6>(0,0,0,0,0,0);
I_lb = getMomentOfInertia(cg_lb, tb, rho_lb);

Array<T,3> a_lb = Array<T,3>(0,0,0);
a_lb = previous.a_lb + f_lb / mass_lb;

Array<T,3> v_lb = Array<T,3>(0,0,0);
v_lb = previous.v_lb + a_lb * (T)1.0;

Array<T,3> ds_lb = Array<T,3>(0,0,0);
Array<T,3> ds_a = (T)0.5 * a_lb * (T)1.0 * (T)1.0;
Array<T,3> ds_v = previous.v_lb * (T)1.0;
ds_lb = ds_v + ds_a;

Array<T,3> alpha_lb = Array<T,3>(0,0,0);
//alpha_lb = previous.alpha_lb + getAlpha(torque_lb, I_lb);

Array<T,3> omega_lb = Array<T,3>(0,0,0);
//omega_lb = previous.omega_lb + alpha_lb * (T)1.0;

Array<T,3> dtheta_lb = Array<T,3>(0,0,0);
//dtheta_lb = previous.omega_lb * (T)1.0 + (T)0.5 * alpha_lb *
```

```
        (T)1.0 * (T)1.0;

    std::vector<Array<T,3> > newVertices;
    newVertices.resize(n);
    newVertices.reserve(n);

    verticesVelocity.clear();
    verticesVelocity.resize(n);
    verticesVelocity.reserve(n);
/*
    TriangleSet<T> simple =
        *triangleSet.toTriangleSet(Constants<T>::precision);
    Array<T,3> axis = Array<T,3>(1,0,0);
    simple.rotateAtOrigin(axis, dtheta_lb[0]);
    axis = Array<T,3>(0,1,0);
    simple.rotateAtOrigin(axis, dtheta_lb[1]);
    axis = Array<T,3>(0,0,1);
    simple.rotateAtOrigin(axis, dtheta_lb[2]);

    simple.translate(ds_lb);

    triangleSet = ConnectedTriangleSet<T>(simple);
*/
    T vv = 0;
    Array<T,3> maxVV_lb = Array<T,3>(0,0,0);
    for(plint i = 0; i < n; i++){
        //pcout << "old Vertex= " << array_string(oldVertices[i]);
        newVertices[i] = Array<T,3>(0,0,0);
        newVertices[i] = getRotation(oldVertices[i],cg_lb,dtheta_lb)
            ;
        newVertices[i] += ds_lb;
        //newVertices[i] = triangleSet.getVertex(i);
        if(moves > 2){if(outOfBounds(domain, newVertices[i])){ stop
            = true; }}
        //pcout << " new Vertex= " << array_string(newVertices[i]);
        verticesVelocity[i] = Array<T,3>(0,0,0);
        verticesVelocity[i] = getTotalVelocity(oldVertices[i],cg_lb,
            omega_lb,v_lb);
        T abs = std::pow(std::pow(verticesVelocity[i][0],2)
                +std::pow(verticesVelocity[i][0],2)
                +std::pow(verticesVelocity[i][0],2),0.5);
        if(abs > vv){ maxVV_lb = verticesVelocity[i]; vv = abs; }
        //pcout << " Vertex velocity=  "<<
            array_string(verticesVelocity[i]) << std::endl;
    }

    tb->getMesh().translate(ds_lb);

    const T PI = std::acos(-1);
    if(dtheta_lb[1]<0 && dtheta_lb[1] > -PI){ dtheta_lb[1] += PI; }
    if(dtheta_lb[1]>PI && dtheta_lb[1]< 2*PI){ dtheta_lb[1] -= PI; }
    tb->getMesh().rotate(dtheta_lb[2], dtheta_lb[1], dtheta_lb[0]);

    cg_lb = getCG(newVertices);

    previous.v_lb = v_lb;
    previous.a_lb = a_lb;
    previous.omega_lb = omega_lb;
    previous.alpha_lb = alpha_lb;
```

```
Array<T,3> cg = cg_lb *dx;
Array<T,3> ds = ds_lb * dx;
Array<T,3> dtheta = dtheta_lb*dx;

T v_conv = dx/dt;
Array<T,3> v = v_lb*v_conv;
Array<T,3> omega = omega_lb*v_conv;
Array<T,3> maxVV = maxVV_lb *v_conv;

T a_conv = dx / dt2;
Array<T,3> a = a_lb*a_conv;
Array<T,3> alpha = alpha_lb*a_conv;

T dx4 = dx*dx*dx*dx;
T f_conv = dx4 / dt2;
Array<T,3> f =  f_lb*f_conv;
Array<T,3> t = torque_lb*f_conv*dx;


#ifdef PLB_DEBUG
    pcout << "[DEBUG] Obstacle Domain= " << box_string(getDomain
        (tb)) << std::endl;
    pcout << " "<< std::endl;
    pcout << "[DEBUG] Kinematics in Dimensionless Units" <<
        std::endl;
    pcout << "[DEBUG]
        ------------------------------------------------"<<std::
        endl;
    pcout << "[DEBUG] Location= "<< array_string(cg_lb) <<std::
        endl;
    pcout << "[DEBUG] Mass= " << safe_string(mass_lb) << std::
        endl;
    pcout << "[DEBUG]
        ------------------------------------------------"<<std::
        endl;
    pcout << "[DEBUG] Translation= "<< array_string(ds_lb) <<
        std::endl;
    pcout << "[DEBUG] Rotation= " <<array_string(dtheta_lb) <<
        std::endl;
    pcout << "[DEBUG]
        ------------------------------------------------"<<std::
        endl;
    pcout << "[DEBUG] Velocity= "<< array_string(v_lb) <<std::
        endl;
    pcout << "[DEBUG] Rotational= "<< array_string(omega_lb) <<
        std::endl;
    pcout << "[DEBUG]
        ------------------------------------------------"<<std::
        endl;
    pcout << "[DEBUG] Acceleration= "<< array_string(a_lb) <<
        std::endl;
    pcout << "[DEBUG] Rotational= "<< array_string(alpha_lb) <<
        std::endl;
    pcout << "[DEBUG]
        ------------------------------------------------"<<std::
        endl;
    pcout << "[DEBUG] Force= "<< array_string(f_lb) <<std::endl;
    pcout << "[DEBUG] Torque= "<< array_string(torque_lb) <<
        std::endl;
    pcout << "[DEBUG]
```

```
                ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Max Vertex Total Velocity= "<<
                    array_string(maxVV_lb) <<std::endl;
                pcout << " " << std::endl;
                pcout << "[DEBUG] Kinematics in Physical Units" << std::endl
                    ;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Location= "<< array_string(cg) <<std::endl
                    ;
                pcout << "[DEBUG] Mass= " << safe_string(mass) << std::endl;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Translation= "<< array_string(ds) <<std::
                    endl;
                pcout << "[DEBUG] Rotation= " <<array_string(dtheta) <<
                    std::endl;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Velocity= "<< array_string(v) <<std::endl;
                pcout << "[DEBUG] Rotational= "<< array_string(omega) <<
                    std::endl;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Acceleration= "<< array_string(a) <<std::
                    endl;
                pcout << "[DEBUG] Rotational= "<< array_string(alpha) <<
                    std::endl;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Force= "<< array_string(f) <<std::endl;
                pcout << "[DEBUG] Torque= "<< array_string(t) <<std::endl;
                pcout << "[DEBUG]
                    ----------------------------------------------"<<std::
                    endl;
                pcout << "[DEBUG] Max Vertex Total Velocity= "<<
                    array_string(maxVV) <<std::endl;
                mesg = "[DEBUG] DONE Updating SurfaceVelocity";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            time.push_back(time_lb);
            forceList.push_back(f);
            acceleration.push_back(a);
            velocity.push_back(v);
            location.push_back(cg);
            angular_acceleration.push_back(alpha);
            angular_velocity.push_back(omega);
            moves++;
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return stop;
    }
```

```
} // NAMESPACE PLB

#endif // VELOCITY_HH
```

A.11. Normal

```
#ifndef NORMAL_H
#define NORMAL_H

namespace plb{

template<typename T>
class SurfaceNormal{
public:
    static int objCount;
    SurfaceNormal();

    Array<T,3> operator()(const pluint& id);

    void update(const TriangleBoundary3D<T>* tb);

private:
    static std::vector<Array<T,3> > normals;
    static bool master;
};


// Initializers
template<typename T>
int SurfaceNormal<T>::objCount= 0;

template<typename T>
std::vector<Array<T,3> > SurfaceNormal<T>::normals;

template<typename T>
bool SurfaceNormal<T>::master= false;

}

#endif
```

normal.hh 07/12/2016, 13:28

```
#ifndef NORMAL_HH
#define NORMAL_HH

namespace plb{

    template<typename T>
    SurfaceNormal<T>::SurfaceNormal()
    {
        if(objCount == 0)
        {
            master = global::mpi().isMainProcessor();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Constructing SurfaceNormal";
                if(master){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            objCount++;
        }
        else
        {
            std::string ex = "Static Class Surface Normal already defined";
            std::string line = std::to_string(__LINE__);
            std::string mesg = "[ERROR]: "+ex+" [FILE:"+__FILE__+",LINE:"+
                line+"]";
            global::log(mesg);
            throw std::runtime_error(mesg);
        }
    }

    template<typename T>
    Array<T,3> SurfaceNormal<T>::operator()(const pluint& id)
    {
        return normals[id];
    }

    template<typename T>
    void SurfaceNormal<T>::update(const TriangleBoundary3D<T>* tb){
        plint numVertices = tb->getMesh().getNumVertices();
        normals.clear();
        normals.resize(numVertices);
        normals.reserve(numVertices);
        const bool weightedArea = false;
        for (plint iVertex = 0; iVertex < numVertices; iVertex++){
            normals[iVertex] = tb->getMesh().computeVertexNormal(iVertex,
                weightedArea);
        }
    }

} // NAMESPACE PLB

#endif // VELOCITY_HH
```

A.12. MPI Data Manager

```
#ifndef MPI_DATA_MANAGER_H
#define MPI_DATA_MANAGER_H

#include <palabos3D.h>
#include "myheaders3D.h"

#ifdef PLB_MPI_PARALLEL
#include <mpi.h>
#include <vector>
#include <string>
#endif

namespace plb{
    namespace global{

    #ifdef PLB_MPI_PARALLEL
    class MpiDataManager{
    public:
        template<typename T>
        void sendScalarField3D(const ScalarField3D<T>& field, const Box3D&
            fromDomain);

        template<typename T>
        void receiveScalarField3D(ScalarField3D<T>& field, const Box3D&
            fromDomain,const int& fromId) const;

        template<typename T>
        TriangleSet<T> receiveTriangleSet();

        template<typename T>
        void sendTriangleSet(const TriangleSet<T>& triangles);

        void sendReceiveDomains(const bool& master, std::vector<Box3D>&
            mpiDomains);

        std::vector<Box3D> splitDomains(const Box3D& domain);

        MpiDataManager& mpiData(){static MpiDataManager instance; return
            instance;}
    private:
        void checkDomain(int rank, Box3D domain, const int& line);
        Array<int,2> checkIfCrossed(const int& fMin, const int& fMax, const
            int& n);
        MpiDataManager();
        ~MpiDataManager();
    friend MpiDataManager& mpiData();
    };
    #endif

    #ifndef PLB_MPI_PARALLEL
    class MpiDataManager{
    public:
        template<typename T>
        void sendScalarField3D(const ScalarField3D<T>& field, const Box3D&
            fromDomain){}

        template<typename T>
        void receiveScalarField3D(ScalarField3D<T>& field, const Box3D&
            fromDomain, const int& fromId) const{}
```

```
        template<typename T>
        TriangleSet<T> receiveTriangleSet(){}

        template<typename T>
        void sendTriangleSet(const TriangleSet<T>& triangles){}

        void sendReceiveDomains(const bool& master, std::vector<Box3D>&
            mpiDomains){}

        void splitDomains(const Box3D& domain){ }

        MpiDataManager& mpiData(){static MpiDataManager instance; return
            instance;}
    private:
        void checkDomain(int rank, Box3D domain, const int& line){};
        void checkIfCrossed(const int& fMin, const int& fMax, const int& n){
            }
        MpiDataManager();
        ~MpiDataManager();
    friend MpiDataManager& mpiData();
    };
    #endif

    inline MpiDataManager& mpiData(){static MpiDataManager instance; return
        instance;}

    }  // namespace global
}  // namespace plb


#endif  // MPI_DATA_MANAGER_H
```

mpiDataManager.hh 07/12/2016, 13:51

```
// Include Guard
#ifndef mpiDataManager_hh
#define mpiDataManager_hh

#ifdef PLB_MPI_PARALLEL
#include "mpiDataManager.h"

#include <palabos3D.hh>
#include "myheaders3D.hh"

#include <string>
#include <exception>

namespace plb{
    namespace global{

    MpiDataManager::MpiDataManager(){}

    MpiDataManager::~MpiDataManager(){}

    inline std::string domain_string(int rank, Box3D domain){
            std::string err_str("Rank= ");
            err_str.append(std::to_string(rank));
            err_str.append(" Domain=[");
            err_str.append(std::to_string(domain.x0));
            err_str.append(",");
            err_str.append(std::to_string(domain.x1));
            err_str.append("][");
            err_str.append(std::to_string(domain.y0));
            err_str.append(",");
            err_str.append(std::to_string(domain.y1));
            err_str.append("][");
            err_str.append(std::to_string(domain.z0));
            err_str.append(",");
            err_str.append(std::to_string(domain.z1));
            err_str.append("]");
            return err_str;
    }

    inline bool domain_empty(Box3D domain){
        return (domain.x0 == 0 && domain.x1 == 0 && domain.y0 == 0 && domain
            .y1 == 0 && domain.z0 ==0 && domain.z1 == 0);
    }

    void MpiDataManager::checkDomain(int rank, Box3D domain,const int& line)
        {
        if((domain.x1 <= domain.x0) || (domain.y1 <= domain.y0) || (domain.
            z1 <= domain.z0)){
            std::string err_str("Rank= ");
            err_str.append(std::to_string(rank));
            err_str.append(" Domain=[");
            err_str.append(std::to_string(domain.x0));
            err_str.append(",");
            err_str.append(std::to_string(domain.x1));
            err_str.append("][");
            err_str.append(std::to_string(domain.y0));
            err_str.append(",");
            err_str.append(std::to_string(domain.y1));
            err_str.append("][");
            err_str.append(std::to_string(domain.z0));
```

```
            err_str.append(",");
            err_str.append(std::to_string(domain.z1));
            err_str.append("]");
            err_str.append(" LINE:");
            err_str.append(std::to_string(line));
            throw std::runtime_error(err_str);
        }
    }

    template<>
    void MpiDataManager::sendScalarField3D<int>(const ScalarField3D<int>&
        field, const Box3D& fromDomain){
        try{
            PLB_PRECONDITION( contained(fromDomain, field.getBoundingBox())
                );
            const pluint nDataPacks = fromDomain.nCells();
            if(nDataPacks==0){return;}
            if(domain_empty(fromDomain)){return;}
            // Create mpi variables
            const int count=4;
            const int rank = mpi().getRank();
            const int nprocs = mpi().getSize();
            #ifdef PLB_DEBUG
                bool main = global::mpi().isMainProcessor();
                std::string mesg = "[DEBUG] Rank "+ std::to_string(rank) + "
                    Trying to Send "+std::to_string(nDataPacks) + "
                    DataPacks";
                if(main){std::cout << mesg << std::endl;}
                global::log(mesg);
            #endif
            plint Xmin = 0; plint Xmax = 0; plint Ymin = 0; plint Ymax = 0;
                plint Zmin = 0; plint Zmax = 0;
            checkDomain(rank,fromDomain,__LINE__);
            if(fromDomain.x0 < fromDomain.x1){ Xmin = fromDomain.x0; Xmax =
                fromDomain.x1;}
            else{ Xmin = fromDomain.x1; Xmax = fromDomain.x0; }
            if(fromDomain.y0 < fromDomain.y1){ Ymin = fromDomain.y0; Ymax =
                fromDomain.y1;}
            else{ Ymin = fromDomain.y1; Ymax = fromDomain.y0; }
            if(fromDomain.z0 < fromDomain.z1){ Zmin = fromDomain.z0; Zmax =
                fromDomain.z1;}
            else{ Zmin = fromDomain.z1; Zmax = fromDomain.z0; }

            int package = 0;
            for (plint iX=Xmin; iX<=Xmax; ++iX) {
                for (plint iY=Ymin; iY<=Ymax; ++iY) {
                    for (plint iZ=Zmin; iZ<=Zmax; ++iZ) {
                        // Create a buffer for the data
                        int* sendBuffer = new int[count];
                        // Sync mpi processes
                        // mpi().barrier();
                        // Fill the buffer
                        sendBuffer[0] = iX; sendBuffer[1]= iY; sendBuffer[2]
                            = iZ; sendBuffer[3]=field.get(iX,iY,iZ);
                        // Send the data to all other processes
                        mpi().barrier();
                        for(int n = 0; n<nprocs; n++){ if(n != rank){ mpi().
                            send(sendBuffer,4,n,package); } }
                        // Clear the buffer
                        delete[] sendBuffer;
```

```
                    package++;
                }
            }
        }
        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Rank "+ std::to_string(rank) + " Done
                Sending "+std::to_string(nDataPacks) + " DataPacks";
            if(main){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<>
void MpiDataManager::receiveScalarField3D<int>(ScalarField3D<int>& field
    , const Box3D& fromDomain, const int& fromId) const{
    try{
        //PLB_PRECONDITION( contained(fromDomain,
            field.getBoundingBox()) );
        const int count = 4;
        const int rank = mpi().getRank();
        if(rank == fromId){return;}
        if(domain_empty(fromDomain)){return;}
        const pluint nDataPacks = fromDomain.nCells();
        int n = 0;
        #ifdef PLB_DEBUG
            bool main = global::mpi().isMainProcessor();
            std::string mesg = "[DEBUG] Rank "+ std::to_string(rank)+"
                Trying to Receive "+std::to_string(nDataPacks)+
            " DataPacks from Rank "+std::to_string(fromId);
            if(main){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
        while(n < nDataPacks){
            // Create a buffer for the data
            int* recvBuffer = new int[count];
            // Sync the mpi processes
            mpi().barrier();
            // Receive the Data
            mpi().receive(recvBuffer, count, fromId, n);
            // Put the data into the ScalarField
            if(sizeof(recvBuffer[3]) <= sizeof(int)){
                field.get(recvBuffer[0], recvBuffer[1], recvBuffer[2]) =
                    (int)recvBuffer[3];
            }
            else{ throw std::overflow_error("INT OVERFLOW "+std::
                to_string(__LINE__));}
            // clear the Buffer
            delete[] recvBuffer;
            n++;
        }
        #ifdef PLB_DEBUG
            mesg = "[DEBUG] Rank "+ std::to_string(rank)+" Done
                Receiving "+std::to_string(nDataPacks)+
            " DataPacks from Rank "+std::to_string(fromId);
            if(main){std::cout << mesg << std::endl;}
            global::log(mesg);
        #endif
```

```
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

template<>
TriangleSet<double> MpiDataManager::receiveTriangleSet<double>(){
    TriangleSet<double> triangles;
    try{
        if(!mpi().isMainProcessor()){
            const int rank = mpi().getRank();
            if(rank == 0){ throw std::runtime_error("Error in
                receiveTriangleSet, Master should not receive");}
            #ifdef PLB_DEBUG
                std::string mesg ="[DEBUG] Rank "+std::to_string(rank)+"
                    is trying to receive a triangleSet";
                std::cout << mesg << std::endl;
                global::log(mesg);
            #endif
            std::vector<Array<Array<double,3>,3> > list;
            plint* buffer = new plint[1];
            mpi().receive(buffer, 1, 0);
            plint nTriangles = *buffer;
            delete[] buffer;
            for(int i=0; i<nTriangles; i++){
                Array<Array<double,3>,3> triangle;
                for(int p =0; p<3; p++){
                    double* recvBuffer = new double[3];
                    Array<double, 3> point;
                    mpi().receive(recvBuffer,3,0);
                    point[0] = recvBuffer[0]; point[1] = recvBuffer[1];
                        point[2] = recvBuffer[2];
                    triangle[p] = point;
                    delete[] recvBuffer;
                }
                list.push_back(triangle);
            }
            triangles = TriangleSet<double>(list, DBL);
            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Rank "+std::to_string(rank)+" received a
                    triangleSet with "+std::to_string(list.size())
                +" Triangles";
                std::cout << mesg << std::endl;
                global::log(mesg);
            #endif
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return triangles;
}

template<>
void MpiDataManager::sendTriangleSet(const TriangleSet<double>&
    triangles){
    try{
        if(mpi().isMainProcessor()){
            const int rank = mpi().getRank();
            if(rank != 0){ throw std::runtime_error("Error in
                sendTriangleSet, process is not Master");}
```

```
            std::vector<Array<Array<double,3>,3> > list = triangles.
                getTriangles();
            plint nTriangles = list.size();
            #ifdef PLB_DEBUG
                std::string mesg = "[DEBUG] Master is trying to send a
                    triangleSet with "+ std::to_string(nTriangles)+"
                    Triangles";
                std::cout << mesg << std::endl;
                global::log(mesg);
            #endif
            const int nprocs = mpi().getSize();
            plint* buffer = new plint[1];
            buffer[0] = nTriangles;
            for(int n = 0; n<nprocs; n++){ if(n != rank){ mpi().send
                (buffer,1,n); } }
            delete[] buffer;
            for(int i=0; i<nTriangles; i++){
                Array<Array<double,3>,3> triangle = list[i];
                for(int p = 0; p<3; p++){
                    double* sendBuffer = new double[3];
                    Array<double,3> point = triangle[p];
                    sendBuffer[0] = point[0]; sendBuffer[1]=point[1];
                        sendBuffer[2]=point[2];
                    for(int n = 0; n<nprocs; n++){ if(n != rank){ mpi().
                        send(sendBuffer,3,n); } }
                    delete[] sendBuffer;
                }
            }
            #ifdef PLB_DEBUG
                mesg = "[DEBUG] Master has sent a triangleSet with "+
                    std::to_string(nTriangles)+" Triangles";
                std::cout << mesg << std::endl;
                global::log(mesg);
            #endif
        }
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

Array<int,2> MpiDataManager::checkIfCrossed(const int& fMin, const int&
    fMax, const int& n){
    Array<int,2> boundary;
    try{
        if(fMin == fMax){ throw std::runtime_error("Exception in
            MpiDataManager::checkIfCrossed domain limits are equal");}
        int min = 0; int max=0;
        if(fMin<0){ min = 0; }
        else{
            if(fMin<fMax){
                min = fMin;
            }
            else{
                min = fMax;
            }
        }
        if(fMax >= n){
            max= n;
        }
        else{
```

```
            if(fMax > fMin){
                max = fMax;
            }
            else{
                max = fMin;
            }
        }
        boundary[0] = min;
        boundary[1] = max;
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
    return boundary;
}

void MpiDataManager::sendReceiveDomains(const bool& master, std::vector<
    Box3D>& mpiDomains){
try{
    const int nprocs = mpi().getSize();
    const int count = 6;
    for(int i = 0; i<nprocs; i++){
        int* buffer = new int[count];
        if(master){
            Box3D domain = mpiDomains[i];
            buffer[0] = domain.x0;
            buffer[1] = domain.x1;
            buffer[2] = domain.y0;
            buffer[3] = domain.y1;
            buffer[4] = domain.z0;
            buffer[5] = domain.z1;
            mpi().barrier();
            for(int s = 0; s < nprocs; s++){ mpi().send(buffer, count, s
                , i); }
        }
        else{
            mpi().barrier();
            mpi().receive(buffer, count, 0, i);
            Box3D domain(buffer[0],buffer[1],buffer[2],buffer[3],buffer
                [4],buffer[5]);
            mpiDomains.push_back(domain);
        }
    }
}
catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
    __LINE__);}
}

std::vector<Box3D> MpiDataManager::splitDomains(const Box3D& domain){
    std::vector<Box3D> mpiDomains;
    try{
        const bool master = global::mpi().isMainProcessor();
        if(master){
            int minX, maxX, minY, maxY, minZ, maxZ;
            Array<int,2> boundary;
            boundary = checkIfCrossed(domain.x0, domain.x1, domain.getNx
                ());
            minX = boundary[0]; maxX = boundary[1];
            boundary = checkIfCrossed(domain.y0, domain.y1, domain.getNy
                ());
            minY = boundary[0]; maxY = boundary[1];
```

```
boundary = checkIfCrossed(domain.z0, domain.z1, domain.getNz
    ());
minZ = boundary[0]; maxZ = boundary[1];
const int nproc = mpi().getSize();
// mpiDomains.resize(nproc);
int nSide = std::cbrt(nproc);
if(nSide == 0){ throw std::runtime_error("Qubic Root of
    nprocs failed");}
plint xdif, xrem, ydif, yrem, zdif, zrem;
bool mpiExcess = false;
if(maxX - minX < nSide){ xdif = 1; mpiExcess = true; }
else{ xdif = floor((maxX-minX)/nSide); }
if(maxY - minY < nSide){ ydif = 1; mpiExcess = true; }
else{ ydif = floor((maxY-minY)/nSide); }
if(maxZ - minZ < nSide){ zdif = 1; mpiExcess = true; }
else{ zdif = floor((maxZ-minZ)/nSide); }
if((maxX-minX) % nSide !=0){ xrem = (maxX-minX) % nSide; }
    else{xrem = 0;}
if((maxY-minY) % nSide !=0){ yrem = (maxY-minY) % nSide; }
    else{yrem = 0;}
if((maxZ-minZ) % nSide !=0){ zrem = (maxZ-minZ) % nSide; }
    else{zrem = 0;}
if(!mpiExcess){ mpiDomains.resize(nproc);}
if(xdif==0 || ydif==0 || zdif==0){
    std::cout << "Rank " << global::mpi().getRank() << "
        Domain [" << minX << "," << maxX << "," << minY <<
        "," << maxY << ","
    << minZ << "," << maxZ << "]" << std::endl;
    throw std::runtime_error("Domain Boundaries are not set
        properly");
}
bool error = false;
std::vector<std::string> error_domains;
int r = 0; int y_last = 0; int x_last = 0; int z_last = 0;
for(int x = 0; x<nSide; x++){
    if(r == nproc){ break;}
    if(x_last == maxX){ break; }
    int x0, x1;
    if(x==0){x0 = minX; x1 = minX + xdif; }
    else{ x0 = x_last + 1; if(x==nSide-1){x1 = maxX;} else{
        x1 = x0 + xdif; } }
    for(int y=0; y<nSide; y++){
        if(y_last == maxY){ break; }
        int y0, y1;
        if(y==0){y0 = minY; y1 = minY + ydif;}
        else{ y0 = y_last + 1; if(y==nSide-1){y1 = maxY;}
            else{ y1 = y0 + ydif;} }
        for(int z=0; z<nSide; z++){
            if(z_last == maxZ){ break; }
            int z0, z1;
            if(z==0){z0 = minZ; z1 = minZ + zdif; }
            else{ z0 = z_last + 1; if(z==nSide-1){z1 = maxZ;
                } else{ z1 = z0 + zdif; } }
            x_last = x1; y_last = y1; z_last = z1;
            if(r == nproc){ break;}
            Box3D rankDomain(x0,x1,y0,y1,z0,z1);
            checkDomain(r, rankDomain, __LINE__);
            if(mpiExcess){mpiDomains.push_back(rankDomain);}
            else{mpiDomains[r] = rankDomain;}
            r++;
```

```
                        }
                    }
                }
                if(mpiExcess){
                    int missing = nproc - mpiDomains.size();
                    for(int i = 0; i < missing; i++){
                        Box3D empty(0,0,0,0,0,0);
                        mpiDomains.push_back(empty);
                    }
                }
                #ifdef PLB_DEBUG
                    for(int i = 0; i<nproc; i++){
                        std::string domain = domain_string(i, mpiDomains[i])
                            ;
                        global::log(domain);
                        if(master){ std::cout << domain << std::endl;}
                    }
                #endif
                if(mpiDomains.size() != nproc){ throw
                    std::runtime_error("One or more mpi Domains where not
                    initialized."); }
            }
            sendReceiveDomains(master,mpiDomains);
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        }
        return mpiDomains;
    }

    } // namespace global
} // namespace plb
#endif
#endif
```

A.13. Exceptions

```
#ifndef LBMEXCEPTIONS_H
#define LBMEXCEPTIONS_H

#include <exception>

class MachException: public std::exception{
  virtual const char* what() const throw()
  {
    return "Compressibility Error too Large. Lower your u0lb in
        parameters.xml section LBM";
  }
}machEx;

class LocalMachException: public std::exception{
  virtual const char* what() const throw()
  {
    return "Local Compressibility Error too Large. Do one of the
        following...\n"
    "1. Decrease u0lb \n"
    "2. Decrease the max Grid refinement Level \n"
    "in parameters.xml \n";
  }
}localMachEx;

class SuperException: public std::exception{
  virtual const char* what() const throw()
  {
    return "The model does not support supersonic flows. Do one of the
        following...\n"
    "1. Increase u0lb \n"
    "2. Decrease the reference Resolution \n"
    "3. Decrease the max Grid refinement Level \n"
    "in parameters.xml \n";
  }
}superEx;

class MarginException: public std::exception{
  virtual const char* what() const throw()
  {
    return "Margin cannot be smaller then borderWidth, please correct the
        xml file";
  }
}marginEx;

class ResolutionException: public std::exception{
  virtual const char* what() const throw()
  {
    return "The resolution cannot be zero, please correct the xml file";
  }
}resolEx;
#endif // define LBMEXCEPTIONS_HH
```

debug.h 07/12/2016, 13:53

A.14. Debug

```
#ifndef DEBUG_H
#define DEBUG_H

// C++ INCLUDES
#include <execinfo.h> // For complete stacktrace on exception see GNU libc
    manual
#include <unistd.h> // For gethostname
#include <sys/types.h> // For getpid
#include <iostream>
#include <sstream>
#include <signal.h>
#include <string>
#include <sys/time.h>
#include <sys/resource.h>



// Palabos INCLUDES
#include <core/plbLogFiles.h>

namespace plb{

void printTrace (void){         // Obtain a stacktrace and print it to
    stdout.
try{
    int length = 100;
    void *array[length];
    size_t size;

    //char **strings;
    //int i;
    size = backtrace(array, length);

    fprintf(stderr, "[ERROR]: EXCEPTION STACKTRACE \n");
    backtrace_symbols_fd(array, size, STDERR_FILENO);

    global::log("[ERROR]: EXCEPTION STACKTRACE \n");

    void* const* buffer = nullptr;
    char** trace;
    trace = backtrace_symbols(buffer, size);
    for(int i = 0; i<length; i++){
        global::log(std::to_string(**trace));
    }
    free(trace);
}
catch(const std::exception& e){
    std::string ex = e.what();
    std::string line = std::to_string(__LINE__);
    global::log().entry("[STACKTRACE]: "+ex+" [FILE:"+__FILE__+",LINE:"+line
        +"]");
    throw e; }
}

void exHandler(const std::exception& e, const std::string& file, const std::
    string& function, const int& line){
    std::string location = "[FILE: "+file+", FUNC: "+function+", LINE: "+
        std::to_string(line)+"]";
    std::string ex = e.what();
    std::string mesg = "[EXCEPTION] "+ ex + location;
```

```
        global::log(mesg);
        std::cerr << mesg << std::endl;
        printTrace();
        throw e;
    }

    void memUsage(){
        try{
            struct rusage usage;
            int answer = getrusage(RUSAGE_SELF, &usage);
            if(answer == 0){
                std::string mesg;
                mesg = std::to_string(usage.ru_utime.tv_sec) + " user CPU time
                    used in seconds";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_stime.tv_sec) + " system CPU time
                    used in seconds";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_maxrss) + " maximum resident set
                    size";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_ixrss) + " integral shared memory
                    size";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_idrss) + " integral unshared data
                    size";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_isrss) + " integral unshared
                    stack size";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_minflt) + " page reclaims (soft
                    page faults)";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_majflt) + " page faults (hard
                    page faults)";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_nswap) + " swaps";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_inblock) + " block input
                    operations";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_oublock) + " block output
                    operations";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_msgsnd) + "IPC messages sent";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_msgrcv) + "IPC messages received"
                    ;
```

```
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_nsignals) + " signals received";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_nvcsw) + " voluntary context
                    switches";
                global::log(mesg);
                pcout << mesg << std::endl;
                mesg = std::to_string(usage.ru_nivcsw) + " involuntary context
                    switches";
                global::log(mesg);
                pcout << mesg << std::endl;
            }
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }

    void sigHandler(int sig) {
        try{
            int length = 100;
            void *array[length];
            size_t size;

            // get void*'s for all entries on the stack
            size = backtrace(array, length);

            std::string mesg = "";
            if(sig > 0 && sig < 32){
            switch(sig){
                case 1: mesg = "[ERROR]: (Signal Hangup) Report that user's
                    terminal is disconnected.\n Signal used to report the
                    termination of the controlling process. \n";
                case 2: mesg = "[ERROR]: (Signal Interrupt) Program interrupted
                    \n";
                case 3: mesg = "[ERROR]: (Signal Quit) Terminate process and
                    generate core dump \n";
                case 4: mesg = "[ERROR]: (Signal Illegal Instruction) Generally
                    indicates that the executable file \n is corrupted or use of
                    data where a pointer to a function was expected. \n";
                case 5: mesg = "[ERROR]: (Signal Trace Trap) \n";
                case 6: mesg = "[ERROR]: (Signal Abort) Process detects error
                    and reports by calling abort. \n";
                case 7: mesg = "[ERROR]: (Signal BUS error) Indicates an access
                    to an invalid address. \n";
                case 8: mesg = "[ERROR]: (Signal Floating-Point Exception)
                    Erroneous arithmetic operation, \n such as zero divide or an
                    operation resulting in overflow (not necessarily with a
                    floating-point operation). \n";
                case 9: mesg = "[ERROR]: (Signal Kill) Cause immediate
                    termination \n";
                case 10: mesg = "[ERROR]: (Signal Userdefined 1) \n";
                case 11: mesg = "[ERROR]: (Signal Segmentation Violation)
                    Invalid access to storage: \n When a program tries to read
                    or write outside the memory it has allocated. \n";
                case 12: mesg = "[ERROR]: (Signal Userdefined 2) \n";
                case 13: mesg = "[ERROR]: (Signal Broken Pipe) Error condition
                    like trying to write to a socket which is not connected. \n"
                    ;
```

```
            case 14: mesg = "[ERROR]: (Signal Alarm Clock) Alarm clock
                (POSIX) Indicates expiration of a timer. Used by the alarm()
                function. \n";
            case 15: mesg = "[ERROR]: (Signal Termination) This signal can
                be blocked, handled, and ignored. Generated by kill command.
                \n";
            case 16:  mesg = "[ERROR]: (Signal Stack fault) \n";
            case 17: mesg = "[ERROR]: (Signal Child status has changed)
                Signal sent to parent process \n whenever one of its child
                processes terminates or stops. \n";
            case 18: mesg = "[ERROR]: (Signal   Continue) Signal sent to
                process to make it continue. \n";
            case 19: mesg = "[ERROR]: (Signal   Stop, unblockable) Stop a
                process. This signal cannot be handled, ignored, or blocked.
                \n";
            case 20: mesg = "[ERROR]: (Signal Keyboard stop) Interactive
                stop signal. This signal can be handled and ignored. (ctrl-
                z) \n";
            case 21: mesg = "[ERROR]: (Signal Background read from tty) \n";
            case 22: mesg = "[ERROR]: (Signal Background write to tty) \n";
            case 23: mesg = "[ERROR]: (Signal Urgent condition on socket)
                Signal sent when urgent or out-of-band data arrives on a
                socket. \n";
            case 24: mesg = "[ERROR]: (Signal CPU limit exceeded) \n";
            case 25: mesg = "[ERROR]: (Signal size limit exceeded) \n";
            case 26: mesg = "[ERROR]: (Signal   Virtual Time Alarm)
                Indicates expiration of a timer. \n";
            case 27: mesg = "[ERROR]: (Signal   Profiling alarm clock)
                Indicates expiration of a timer. Use for code profiling
                facilities. \n";
            case 28: mesg = "[ERROR]: (Signal   Window size change) \n";
            case 29: mesg = "[ERROR]: (Signal I/O now possible) Pollable
                event occurred (System V) \n Signal sent when file
                descriptor is ready to perform I/O (generated by sockets)
                \n";
            case 30: mesg = "[ERROR]: (Signal   Power failure restart) \n";
            case 31: mesg = "[ERROR]: (Signal   Bad system call) \n";
        }
        }
        else{  mesg = "[ERROR]: (Signal "+std::to_string(sig)+") \n";}
        // print out all the frames to stderr
        global::log(mesg);
        fprintf(stderr, mesg.c_str(), sig);

        memUsage();

        backtrace_symbols_fd(array, size, STDERR_FILENO);

        void* const* buffer = nullptr;
        char** trace;
        trace = backtrace_symbols(buffer, size);
        for(int i = 0; i<length; i++){
            global::log(std::to_string(**trace));
        }
        free(trace);
        exit(1);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}
```

```
void installSigHandler(){
    try{
        #ifdef __APPLE__
            signal(SIGVTALRM, sigHandler);
            signal(SIGTSTP, sigHandler);
        #elif __linux__
            signal(SIGSTKFLT, sigHandler);
            signal(SIGPWR, sigHandler);
            signal(SIGVTALRM, sigHandler);
            signal(SIGTSTP, sigHandler);
        #else
            signal(SIGSTP, sigHandler);
            signal(SIGVALRM, sigHandler);
        #endif
        signal(SIGHUP, sigHandler);
        signal(SIGINT, sigHandler);
        signal(SIGQUIT, sigHandler);
        signal(SIGILL, sigHandler);
        signal(SIGTRAP, sigHandler);
        signal(SIGABRT, sigHandler);
        signal(SIGFPE, sigHandler);
        signal(SIGKILL, sigHandler);
        signal(SIGUSR1, sigHandler);
        signal(SIGSEGV, sigHandler);
        signal(SIGUSR2, sigHandler);
        signal(SIGPIPE, sigHandler);
        signal(SIGALRM, sigHandler);
        signal(SIGTERM, sigHandler);
        signal(SIGCHLD, sigHandler);
        signal(SIGCONT, sigHandler);
        signal(SIGSTOP, sigHandler);
        signal(SIGTTIN, sigHandler);
        signal(SIGTTOU, sigHandler);
        signal(SIGURG, sigHandler);
        signal(SIGXCPU, sigHandler);
        signal(SIGXFSZ, sigHandler);
        signal(SIGPROF, sigHandler);
        signal(SIGWINCH, sigHandler);
        signal(SIGIO, sigHandler);
        signal(SIGSYS, sigHandler);
        signal(SIGBUS, sigHandler);
        signal(SIGTRAP, sigHandler);
    }
    catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
        __LINE__);}
}

#ifdef PLB_MPI_PARALLEL
void waitGDB(void){
    try{
        int i = 0;
        char hostname[256];
        gethostname(hostname, sizeof(hostname));
        const int pid = getpid();
        std::string mesg = "PID "+std::to_string(pid)+" on "+hostname+"
            ready for GDB attach";
        std::cout << mesg << std::endl;
        global::log().entry("[GDB]: "+mesg+ " [FILE:"+__FILE__+",LINE:"+
            std::to_string(__LINE__)+"]");
```

```
            fflush(stdout);
            unsigned int seconds;
            seconds = 10;
            while (0 == i){ }
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
    }
    #endif

    template<typename T>
    std::string adr_string(const T& var){
        std::string ans ="";
        try{
            const void * address = static_cast<const void*>(var);
            std::stringstream ss;
            ss << address;
            ans = ss.str();
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return ans;
    }

    std::string safe_string(const T& input){
        std::string ans ="";
        try{
            const T val = (T)input;
            const T min = std::numeric_limits<T>::min();
            const T max = std::numeric_limits<T>::max();
            if(std::isnan(val)){ ans ="NaN";}
            else if(val == 0){ ans ="0"; }
            else if(val>max){ans = "Inf"; }
            else if(val<0){
                T temp = (T)-1.0 * val;
                if(temp > max){ ans = "-Inf"; }
                else{ ans = std::to_string(val);}
            }
            //else if(!val){    ans = "NULL";}
            else{ ans = std::to_string(val);}
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return ans;
    }

    std::string array_string(const Array<T,3>& array){
        std::string ans = "";
        try{
            std::string a1 = safe_string(array[0]);
            std::string a2 = safe_string(array[1]);
            std::string a3 = safe_string(array[2]);
            ans = "["+a1+", "+a2+", "+a3+"]";
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return ans;
    }

    std::string box_string(const Box3D& box){
```

```
        std::string ans = "";
        try{
            std::string a1 = safe_string(box.x0);
            std::string a2 = safe_string(box.x1);
            std::string a3 = safe_string(box.y0);
            std::string a4 = safe_string(box.y1);
            std::string a5 = safe_string(box.z0);
            std::string a6 = safe_string(box.z1);
            ans = "["+a1+", "+a2+"]["+a3+", "+a4+" ]["+a5+", "+a6+"]";
        }
        catch(const std::exception& e){exHandler(e,__FILE__,__FUNCTION__,
            __LINE__);}
        return ans;
    }


    }
    #endif //DEBUG_H
```

A.15. Geo

```
#ifndef GEO_H
#define GEO_H

#include <core/array.h>

namespace plb{

template<typename T>
class Point{
// Default Constructor
public:
    Point();
    Point(const T &_x, const T &_y, const T &_z);
    Point(const Array<T,3> &array);
    Point(const Point &p);
// Default Destructor
    ~Point(){}
// Methods
    T distance(const Point &p);
// Properties
T x;
T y;
T z;
};

template<typename T>
class Triangle{
// Default Constructor
public:
    Triangle(){} // Default constructor that calls the default point
        constructor for a,b and c
    Triangle(const Point<T> &_a, const Point<T> &_b, const Point<T> &_c);
    Triangle(const Array<Array<T,3>,3> &array);
// Default Destructor
    ~Triangle(){};
// Methods
    Point<T> getCentroid();
    double area();
// Properties
    Point<T> a, b, c;
};

template<typename T>
class Pyramid{
// Default Constructor
public:
    Pyramid(){} // Default constructor that calls the default point
        constructor for a,b and c
    Pyramid(const Triangle<T> &b, const Point<T> &t);
// Default Destructor
    ~Pyramid(){};
// Methods
    T volume();
// Properties
    Triangle<T> base;
    Point<T> top;
};
}// Namespace plb

#endif // GEO_H
```

geo.hh 07/12/2016, 13:26

```
#ifndef GEO_HH
#define GEO_HH

#include "geo.h"

namespace plb{

/*
    ////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
 *                      CLASS POINT
 *
     ////////////////////////////////////////////////////////////////////////
     ////////////////////////////////////////////////////////////*/

    template<typename T>
    Point<T>::Point(){
        this->x = 0;
        this->y = 0;
        this->z = 0;
    }

    template<typename T>
    Point<T>::Point(const T &_x, const T &_y, const T &_z):x(_x),y(_y),z(_z)
        {}

    template<typename T>
    Point<T>::Point(const Array<T,3> &array):x(array[0]),y(array[1]),z(array
        [2]){}

    template<typename T>
    Point<T>::Point(const Point &p):x(p.x),y(p.y),z(p.z){}

    template<typename T>
    T Point<T>::distance(const Point &p){
        T d = 0;
        d = sqrt(pow((this->x - p.x),2) + pow((this->y - p.y),2) + pow((this
            ->z - p.z),2));
        return d;
    }

/*
    ////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////
 *                      CLASS TRIANGLE
 *
     ////////////////////////////////////////////////////////////////////////
     ////////////////////////////////////////////////////////////*/

    template<typename T>
    Triangle<T>::Triangle(const Point<T> &_a, const Point<T> &_b, const
        Point<T> &_c):a(_a),b(_b),c(_c){}

    template<typename T>
    Triangle<T>::Triangle(const Array<Array<T,3>,3> &array){
            this->a = Point<T>(array[0]);
            this->b = Point<T>(array[1]);
            this->c = Point<T>(array[2]);
    }
```

```
    template<typename T>
    Point<T> Triangle<T>::getCentroid(){
        Point<T> center(1/3*(this->a.x + this->b.x + this->c.x),
        1/3*(this->a.y + this->b.y + this->c.y), 1/3*(this->a.z + this->b.z
            + this->c.z));
        return center;
    }

    template<typename T>
    double Triangle<T>::area() { // Heron's formula
        double area = 0; double AB=0; double BC=0; double CA=0; double s = 0
            ;
        AB = this->a.distance(this->b);
        BC = this->b.distance(this->c);
        CA = this->c.distance(this->a);
        s = (AB + BC + CA)/2;
        area = sqrt(s*(s-AB)*(s-BC)*(s-CA));
        return area;
    }

/*
    ###########################################################################
    ################################################################
 *                      CLASS PYRAMID
 *
     ###########################################################################
     ################################################################*/

    template<typename T>
    Pyramid<T>::Pyramid(const Triangle<T> &b, const Point<T> &t):base(b),top
        (t){}

    template<typename T>
    T Pyramid<T>::volume() {
        T volume = 0; T area = 0; T height = 0;
        Point<T> centroid = base.getCentroid();
        area = base.area();
        height = centroid.distance(this->top);
        volume = area*height;
        volume = volume/3;
        return volume;
    }

}// namespace plb

#endif // GEO_HH
```

B Simulation Log

```
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing
SurfaceNormal
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing
SurfaceVelocity
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Constants
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Wall
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Obstacle
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Variables
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Output
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Constructing Helper
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Creating Constants
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Done Creating
Constants
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Creating Wall
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] MeshFileName =/home/
jseelen/Palabos/build/input/cylinder.stl
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Domain BEFORE Scaling
[-1.000000, 1.000000][-1.000000, 1.000000 ][0, 5.000000] in physical
units
Sun Nov 13 14:55:50 2016[1,0]<stdout>:[DEBUG] Domain AFTER scaling
[0, 0][0, 0 ][0, 0] in physical units
Sun Nov 13 14:55:52 2016[1,0]<stdout>:The wall surface  has 190466
vertices and 380928 triangles.
Sun Nov 13 14:55:52 2016[1,0]<stdout>:The wall surface has a maximum
triangle edge length of 0.000378734
Sun Nov 13 14:55:52 2016[1,0]<stdout>:dx = 0.0004
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Done Initializing Wall
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Initializing Obstacle
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] MeshFileName =/home/
jseelen/Palabos/build/input/ball.stl
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Domain BEFORE Scaling
[-1.000000, 1.000000][-1.000000, 1.000000 ][-1.000000, 1.000000] in
physical units
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Domain AFTER scaling
[0, 0][0, 0 ][0, 0] in physical units
Sun Nov 13 14:55:52 2016[1,0]<stdout>:The immersed surface  has
24578 vertices and 49152 triangles.
Sun Nov 13 14:55:52 2016[1,0]<stdout>:The immersed surface has a
maximum triangle edge length of 0.000329211
Sun Nov 13 14:55:52 2016[1,0]<stdout>:dx = 0.0004
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] DONE Center=
[0.010006, 0.009992, 0.010014]
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Calculating Volume
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] DONE Volume= 0.000004
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Initializing
SurfaceVelocity
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] DONE Initializing
SurfaceVelocity
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Volume= 0.000004 Mass=
0.014435
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Done Initializing
Obstacle
Sun Nov 13 14:55:52 2016[1,0]<stdout>:SIMULATION START Sun Nov 13
```

```
14:55:52 2016
Sun Nov 13 14:55:52 2016[1,0]<stdout>:
Sun Nov 13 14:55:52 2016[1,0]<stdout>:Min Reynolds = 50 Max Reynolds
= 51
Sun Nov 13 14:55:52 2016[1,0]<stdout>:Min Grid Level = 0 Max Grid
Level = 0
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Updating Parameters
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Reynolds=50.000000
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Grid Level=0.000000
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Resolution=50.000000
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Lattice U=0.005000
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Lattice dt=0.000002
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Lattice dx=0.000400
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Scaling
Factor=125000.000000
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Done Updating
Parameters
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Constructing Main
Lattice
Sun Nov 13 14:55:52 2016[1,0]<stdout>:[DEBUG] Creating Mesh
Sun Nov 13 14:55:53 2016[1,0]<stdout>:[DEBUG] Bounded Cuboid BEFORE
Scaling Lower Left Corner [-0.005000, -0.005000, 0] Upper Right
Corner [0.025000, 0.025000, 0.050000] in physical units
Sun Nov 13 14:55:53 2016[1,0]<stdout>:[DEBUG] Bounded Cuboid AFTER
Scaling Lower Left Corner [-12.500000, -12.500000, 0] Upper Right
Corner [62.500000, 62.500000, 125.000000] in dimensionless units
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Mesh address=
0x19393a0
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Elapsed Time=-0.762030
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Creating Mesh
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Bounded Cuboid BEFORE
Scaling Lower Left Corner [0, 0, 0] Upper Right Corner [0.020000,
0.020000, 0.020000] in physical units
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Bounded Cuboid AFTER
Scaling Lower Left Corner [0, 0, 0] Upper Right Corner [50.000000,
50.000000, 50.000000] in dimensionless units
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Mesh address=
0x55b9620
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.714471
Sun Nov 13 14:55:58 2016[1,0]<stdout>:[DEBUG] Creating Triangle
Boundary
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] TriangleBoundary
address= 0x1937ed0
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Elapsed Time=-0.448444
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Creating Triangle
Boundary
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] TriangleBoundary
address= 0x53305d0
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.413492
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Moving Obstacle to
Start Position
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 44.615956]
```

```
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] DONE Center=
[28.015038, 27.979664, 28.036312]
Sun Nov 13 14:56:00 2016[1,0]<stdout>:[DEBUG] Obstacle Original
Position= [2.000000, 52.000000][2.000000, 52.000000 ][3.000000,
53.000000] Center= [28.015038, 27.979664, 28.036312]
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036312]
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] Wall Domain=
[2.000000, 53.000000][2.000000, 53.000000 ][3.000000, 86.000000]
Center= [27.629576, 28.304823, 44.615956]
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] Obstacle Start
Position= [2.000000, 52.000000][3.000000, 53.000000 ][35.000000,
85.000000] Center= [27.629576, 28.304823, 60.036312]
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] DONE Moving Obstacle
to Start Position
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] Creating Voxelized
Domain
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] TB Address=0x1937ed0
FlowType=3 ExtraLayer=0 Borderwidth= 1 EnvelopeWidth= 1 Blocksize= 1
GridLevel= 0 Dynamic Mesh= 0
Sun Nov 13 14:56:01 2016[1,0]<stdout>:[DEBUG] Processing Generic
Blocks
Sun Nov 13 14:56:31 2016[1,0]<stdout>:[DEBUG] DONE Processing
Generic Blocks
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG] Computing Sparse
VoxelMatrix
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG]      1. setting to
constant
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG]      2. adding layer
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG]      3. computing
sparse blockmanagement
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG] Computing Sparse
Management
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG] DONE Computing Sparse
Management
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG]      4. creating new
MultiScalarField3D
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG]      5. copy non local
Sun Nov 13 14:56:32 2016[1,0]<stdout>:[DEBUG] DONE Computing Sparse
VoxelMatrix
Sun Nov 13 14:56:33 2016[1,0]<stdout>:[DEBUG] VoxelizedDomain
address= 0xc9f4c50
Sun Nov 13 14:56:33 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.045937
Sun Nov 13 14:56:33 2016[1,0]<stdout>:[DEBUG] Creating Voxelized
Domain
Sun Nov 13 14:56:33 2016[1,0]<stdout>:[DEBUG] TB Address=0x53305d0
FlowType=1 ExtraLayer=0 Borderwidth= 1 EnvelopeWidth= 1 Blocksize= 1
GridLevel= 0 Dynamic Mesh= 1
Sun Nov 13 14:56:33 2016[1,0]<stdout>:[DEBUG] Processing Generic
Blocks
Sun Nov 13 14:56:35 2016[1,0]<stdout>:[DEBUG] DONE Processing
Generic Blocks
```

```
Sun Nov 13 14:56:35 2016[1,0]<stdout>:[DEBUG] Computing Sparse
VoxelMatrix
Sun Nov 13 14:56:35 2016[1,0]<stdout>:[DEBUG]      1. setting to
constant
Sun Nov 13 14:56:35 2016[1,0]<stdout>:[DEBUG]      2. adding layer
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]      3. computing
sparse blockmanagement
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Computing Sparse
Management
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] DONE Computing Sparse
Management
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]      4. creating new
MultiScalarField3D
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]      5. copy non local
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] DONE Computing Sparse
VoxelMatrix
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] VoxelizedDomain
address= 0xca19bb0
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.109638
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Joining Lattices
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 14:56:36 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 14:56:36 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 14:56:36 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036312]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
```

```
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Location= [0.011052,
0.011322, 0.024015]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Mass= 0.014435
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000020]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-9.810000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-0.141611]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] DONE Updating
SurfaceVelocity
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Domain Information
Lattice = [0, 56.000000][0, 56.000000 ][0, 89.000000] Obstacle =
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
```

```
and Wall = [2.000000, 53.000000][2.000000, 53.000000 ][3.000000,
86.000000]
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Domain= [0, 56.000000]
[0, 56.000000 ][0, 89.000000] Nx= 57.000000 Ny= 57.000000 Nz=
90.000000
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Done Joining Lattices
time=0.433682
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Creating Boundary
Profile
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Defining Profiles for
380928.000000 surface triangles
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Boundary Profile
address= 0xed364f0
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.075671
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Creating Triangle Flow
Shape
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Triangle Flow Shape
address= 0xd50a720
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.000063
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Creating Model
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Model address=
0xc773e40
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.000068
Sun Nov 13 14:56:36 2016[1,0]<stdout>:[DEBUG] Creating
BoundaryCondition
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] BoundaryCondition
address= 0x509e690
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Elapsed Time=-0.595241
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Creating Boundary
Profile
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Defining Profiles for
49152.000000 surface triangles
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Boundary Profile
address= 0xef8d260
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.007807
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Creating Triangle Flow
Shape
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Triangle Flow Shape
address= 0x509f240
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.000050
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Creating Model
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Model address=
0xb509300
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.000060
Sun Nov 13 15:00:52 2016[1,0]<stdout>:[DEBUG] Creating
BoundaryCondition
Sun Nov 13 15:01:10 2016[1,0]<stdout>:[DEBUG] BoundaryCondition
address= 0xef8d770
Sun Nov 13 15:01:10 2016[1,0]<stdout>:[DEBUG] Elapsed Time=0.422162
Sun Nov 13 15:01:10 2016[1,0]<stdout>:[DEBUG] Initializing Lattice
Sun Nov 13 15:05:37 2016[1,0]<stdout>:[DEBUG] Done Initializing
Lattice time=0.433748
Sun Nov 13 15:05:37 2016[1,0]<stdout>:[DEBUG] Done Constructing Main
Lattice
```

```
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Moving Obstacle
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036312]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 15:10:46 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 15:10:46 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 15:10:46 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036312]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
```

```
Velocity= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Location= [0.011052,
0.011322, 0.024015]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Mass= 0.014435
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000059]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-19.620000]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-0.141611]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:10:46 2016[1,0]<stdout>:[DEBUG] DONE Updating
SurfaceVelocity
Sun Nov 13 15:10:47 2016[1,0]<stdout>:[DEBUG] Updating Immersed Wall
Sun Nov 13 15:10:47 2016[1,0]<stdout>:[DEBUG] DONE Updating Immersed
Wall
Sun Nov 13 15:10:47 2016[1,0]<stdout>:[DEBUG] DONE Moving Obstacle
Sun Nov 13 15:10:47 2016[1,0]<stdout>:[DEBUG] Updating Main Lattice
Sun Nov 13 15:10:47 2016[1,0]<stdout>:[DEBUG] Done Updating Main
Lattice
Sun Nov 13 15:10:47 2016[1,0]<stdout>:N collisions=1
Sun Nov 13 15:10:47 2016[1,0]<stdout>:Grid Level=0N collisions=1
Sun Nov 13 15:10:48 2016[1,0]<stdout>:[DEBUG] Writing VTK
Sun Nov 13 15:10:48 2016[1,0]<stdout>:1
Sun Nov 13 15:10:48 2016[1,0]<stdout>:2
Sun Nov 13 15:10:49 2016[1,0]<stdout>:[DEBUG] Writing Density
Sun Nov 13 15:10:49 2016[1,0]<stdout>:[DEBUG] Done Writing Density
Sun Nov 13 15:10:49 2016[1,0]<stdout>:[DEBUG] Writing Velocity
Sun Nov 13 15:10:51 2016[1,0]<stdout>:[DEBUG] Done Writing Velocity
Sun Nov 13 15:10:51 2016[1,0]<stdout>:[DEBUG] Writing Vorticity
Sun Nov 13 15:10:52 2016[1,0]<stdout>:[DEBUG] Done Writing Vorticity
Sun Nov 13 15:10:52 2016[1,0]<stdout>:[DEBUG] Done Writing VTK
```

```
Sun Nov 13 15:15:38 2016[1,0]<stdout>:[DEBUG] Moving Obstacle
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036312]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 15:15:39 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 15:15:39 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 15:15:39 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036312]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000001]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
```

```
Velocity= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Location= [0.011052,
0.011322, 0.024015]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Mass= 0.014435
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000118]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-29.430000]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-0.141611]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] DONE Updating
SurfaceVelocity
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Updating Immersed Wall
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] DONE Updating Immersed
Wall
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] DONE Moving Obstacle
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Updating Main Lattice
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Done Updating Main
Lattice
Sun Nov 13 15:15:39 2016[1,0]<stdout>:N collisions=2
Sun Nov 13 15:15:39 2016[1,0]<stdout>:Grid Level=0N collisions=2
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Writing VTK
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Writing Density
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Done Writing Density
Sun Nov 13 15:15:39 2016[1,0]<stdout>:[DEBUG] Writing Velocity
Sun Nov 13 15:15:40 2016[1,0]<stdout>:[DEBUG] Done Writing Velocity
Sun Nov 13 15:15:40 2016[1,0]<stdout>:[DEBUG] Writing Vorticity
Sun Nov 13 15:15:40 2016[1,0]<stdout>:[DEBUG] Done Writing Vorticity
Sun Nov 13 15:15:40 2016[1,0]<stdout>:[DEBUG] Done Writing VTK
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Moving Obstacle
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Calculating Center
```

```
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036312]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 15:20:14 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 15:20:14 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 15:20:14 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036311]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000001]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000001]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:
```

```
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Location= [0.011052,
0.011322, 0.024015]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Mass= 0.014435
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000196]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-39.240000]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-0.141611]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] DONE Updating
SurfaceVelocity
Sun Nov 13 15:20:14 2016[1,0]<stdout>:[DEBUG] Updating Immersed Wall
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] DONE Updating Immersed
Wall
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] DONE Moving Obstacle
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Updating Main Lattice
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Done Updating Main
Lattice
Sun Nov 13 15:20:15 2016[1,0]<stdout>:N collisions=3
Sun Nov 13 15:20:15 2016[1,0]<stdout>:Grid Level=0N collisions=3
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Writing VTK
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Writing Density
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Done Writing Density
Sun Nov 13 15:20:15 2016[1,0]<stdout>:[DEBUG] Writing Velocity
Sun Nov 13 15:20:16 2016[1,0]<stdout>:[DEBUG] Done Writing Velocity
Sun Nov 13 15:20:16 2016[1,0]<stdout>:[DEBUG] Writing Vorticity
Sun Nov 13 15:20:16 2016[1,0]<stdout>:[DEBUG] Done Writing Vorticity
Sun Nov 13 15:20:16 2016[1,0]<stdout>:[DEBUG] Done Writing VTK
Sun Nov 13 15:25:19 2016[1,0]<stdout>:[DEBUG] Moving Obstacle
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036311]
```

```
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 15:25:20 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 15:25:20 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 15:25:20 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036310]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000001]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000001]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
```

```
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Location= [0.011052,
0.011322, 0.024015]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Mass= 0.014435
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000294]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-49.050000]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-0.141611]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] DONE Updating
SurfaceVelocity
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Updating Immersed Wall
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] DONE Updating Immersed
Wall
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] DONE Moving Obstacle
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Updating Main Lattice
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Done Updating Main
Lattice
Sun Nov 13 15:25:20 2016[1,0]<stdout>:N collisions=4
Sun Nov 13 15:25:20 2016[1,0]<stdout>:Grid Level=0N collisions=4
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Writing VTK
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Writing Density
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Done Writing Density
Sun Nov 13 15:25:20 2016[1,0]<stdout>:[DEBUG] Writing Velocity
Sun Nov 13 15:25:21 2016[1,0]<stdout>:[DEBUG] Done Writing Velocity
Sun Nov 13 15:25:21 2016[1,0]<stdout>:[DEBUG] Writing Vorticity
Sun Nov 13 15:25:22 2016[1,0]<stdout>:[DEBUG] Done Writing Vorticity
Sun Nov 13 15:25:22 2016[1,0]<stdout>:[DEBUG] Done Writing VTK
Sun Nov 13 15:30:16 2016[1,0]<stdout>:[DEBUG] Moving Obstacle
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Calculating Center
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] DONE Center=
[27.629576, 28.304823, 60.036310]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Updating
SurfaceVelocity
```

```
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Input in Dimensionless
Units
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] FluidForce= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] FluidTorque= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Lattice Domain= [0,
56.000000][0, 56.000000 ][0, 89.000000]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]: Moment of Inertia =
Sun Nov 13 15:30:17 2016[1,0]<stdout>:| 8.23738e+20 4.06686e+14
-3.98736e+15 |
Sun Nov 13 15:30:17 2016[1,0]<stdout>:| 4.06686e+14 8.23759e+20
4.06686e+14 |
Sun Nov 13 15:30:17 2016[1,0]<stdout>:| -3.98736e+15 4.06686e+14
8.23768e+20 |
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Obstacle Domain=
[2.000000, 52.000000][3.000000, 53.000000 ][35.000000, 85.000000]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Kinematics in
Dimensionless Units
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Location= [27.629576,
28.304823, 60.036308]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Mass= 225552105.921554
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Translation= [0, 0,
-0.000002]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Rotation= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Velocity= [0, 0,
-0.000002]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Acceleration= [0, 0,
-0.000001]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Rotational= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Force= [0, 0,
-22.126662]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Torque= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Max Vertex Total
Velocity= [0, 0, 0]
Sun Nov 13 15:30:17 2016[1,0]<stdout>:
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG] Kinematics in Physical
Units
Sun Nov 13 15:30:17 2016[1,0]<stdout>:[DEBUG]
------------------------------------------------
```

Bibliography

[1] 10 2011. URL wiki.palabos.org/models:bc.

[2] O. Beneš, M. Salanne, M. Levesque, and R.J.M. Konings. Physico-chemical properties of the msfr fuel salt. Technical report, European Commission, European Research Area, 11 2013.

[3] P.L. Bhatnagar, E.P. Gross, and M. Krook. A model for collision processes in gases. i. small amplitude processes in charged and neutral one-component systems. *Physical Review*, 94(3):511–525, 1954. doi: 10.1103/PhysRev.94.511. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-26344468007&partnerID=40&md5=e6433257952cd5209a2506c320b6478f. cited By 3796.

[4] Ludwig Boltzmann. Populare schriften, essay 3, address to a formal meeting of the imperial academy of science. In *The second law of thermodynamics*, number 3, 05 1886.

[5] M'hamed Bouzidi, Mouaouia Firdaouss, and Pierre Lallemand. Momentum transfer of a boltzmann-lattice fluid with boundaries. *Physics of Fluids*, 13:3452–3459, 2001.

[6] M. Brovchenko, D. Heuer, E. Merle-Lucotte, M. Allibert, N.Capellan, V. Ghetta, and A. Laureau. Preliminary safety calculations to improve the design of molten salt fast reactor. In *PHYSOR 2012 Advances in Reactor Physics Linking Research, Industry, and Education*, 2012.

[7] Cambridge. Cambridge english dictionary, 09 2016. URL http://dictionary.cambridge.org/dictionary/english/sustainable-energy?a=business-english.

[8] Huihong Cheng, Yanchao Qiao, Chang Liu, Yongbing Li, Bojing Zhu, Yaolin Shi, Dongsheng Sun, Kai Zhang, and Weiren Lin. Extended hybrid pressure and velocity boundary conditions for d3q27 lattice boltzmann model. *Applied Mathematical Modelling*, 36(5):2031–2055, 2012.

[9] Bastien Chopard and Michel Droz. *Cellular Automata of Modeling Physical Systems*. Cambridge University Press, 1998.

[10] Olga Filippova and Dieter Haenel. A novel numerical scheme for reactive flows at low mach numbers. *Computer Physics Communications*, 129(1):267 – 274, 2000. ISSN 0010-4655. doi: http://dx.doi.org/10.1016/S0010-4655(00)00113-2. URL http://www.sciencedirect.com/science/article/pii/S0010465500001132.

[11] *Technology Roadmap Update for Generation IV Nuclear Energy Systems*, January 2014. Generation IV International Forum, OECD Nuclear Energy Agency.

[12] Z. Guo, C. Zheng, and B. Shi. An extrapolation method for boundary conditions in lattice boltzmann method. *Physics of Fluids*, 14(6):2007–2010, 2002. doi: 10.1063/1.1471914. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0036611898&partnerID=40&md5=eaafd9b9814c3f7307b042ce70a7bd37. cited By 358.

[13] K. H. Huebner, E. A. Thornton, and T. D. Byron. *The Finite Element Method for Engineers*. Wiley Interscience, third edition, 1995.

[14] T. Inamuro. Lattice boltzmann methods for moving boundary flows. *Fluid Dynamics Research*, 44(2), 2012. doi: 10.1088/0169-5983/44/2/024001. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-84858055718&partnerID=40&md5=c9b4cc6361ec7a6503e7091b6746d6b9. cited By 17.

[15] H. Lamb. *Hydrodynamics*. Cambridge University Press, 6 edition, 1994.

[16] Jonas Latt. Palabos, cfd, complex physics.

[17] Jonas Lätt. *Hydrodynamic limit of lattice Boltzmann equations*. PhD thesis, Université de Genève, 2007.

[18] Jonas Latt. Choice of units in lattice boltzmann simulations. *LBMethod.org*, 2008.

[19] NERA. Nuclear energy and radiation applications, 09 2016. URL http://www.nera.rst.tudelft.nl.

[20] R.R. Nourgaliev, T.N. Dinh, T.G. Theofanous, and D. Joseph. The lattice boltzmann equation method: Theoretical interpretation, numerics and implications. *International Journal of Multiphase Flow*, 29(1):117–169, 2003. doi: 10.1016/S0301-9322(02)00108-8. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-0037247762&partnerID=40&md5=c581e3530dd5ee40afbea7a7e1b4487e. cited By 248.

[21] Suhas Patankar. *Numerical Heat Transfer and Fluid FLow*. Number ISBN 0891165223. Hemisphere Publishing Corporation, 1980.

[22] D. Arumuga Perumal and Anoop K. Dass. A review on the development of lattice boltzmann computation of macro fluid flows and heat transfer. *Alexandria Engineering Journal*, 54(4):955 – 971, 2015. ISSN 1110-0168. doi: http://dx.doi.org/10.1016/j.aej.2015.07.015. URL http://www.sciencedirect.com/science/article/pii/S1110016815001362.

[23] C.S. Peskin. Flow patterns around heart valves: A numerical method. *Journal of Computational Physics*, 10(2):252–271, 1972. doi: 10.1016/0021-9991(72)90065-4. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-49649145203&partnerID=40&md5=0e81d6f4df843ea00798a65473c893fc. cited By 863.

[24] Rolf Rannacher. *Finite Element Methods for the Incompressible Navier-Stokes Equations*. Institute of Applied Mathematics University of Heidelberg, 08 1999.

[25] M. Rohde. *Extending the Lattice Boltzmann Method*. PhD thesis, Delft University of Technology, 2004.

[26] M. Rohde, J.J. Derksen, and H.E.A. Van Den Akker. Volumetric method for calculating the flow around moving objects in lattice-boltzmann schemes. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 65(056701):056701/1–056701/11, May 2002.

[27] M. Rohde, D. Kandhai, J.J. Derksen, and H.E.A. Van Den Akker. Improved bounce-back methods for no-slip walls in lattice-boltzmann schemes: Theory and simulations. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 67(066703):066703/1–066703/10, June 2003.

[28] M. Rohde, D. Kandhai, J.J. Derksen, and H.E.A. Van Den Akker. A generic, mass conservative local grid refinement technique for lattice-boltzmann schemes. *International Journal for Numerical Methods in Fluids*, 51(4):439–468, June 2006.

[29] M. Rohde, J.J. Derksen, and H.E.A. Van den Akker. An applicability study of advanced lattice-boltzmann techniques for moving, no-slip boundaries and local grid refinement. *Elsevier*, 2007.

[30] Jérôme Serp, Michel Allibert, Ondrej Benes, Sylvie Delpech, Olga Feynberg, Véronique Ghetta, Daniel Heuer, David Holcomb, Victor Ignatievr Ignatiev, Jan Leen Kloosterman, Lelio Luzzi, Elsa Merle-Lucotte, Jan Uhlír, Ritsuo Yoshioka, and Dai Zhimin. The molten salt reactor (msr) in generation iv: Overview and perspectives. *Elsevier*, 2014.

[31] R. Serrano-López, J. Fradera, and S. Cuesta-López. Molten salts database for energy applications. *Chemical Engineering and Processing: Process Intensification*, 09 2013.

[32] A. ten Cate, C.H. Nieuwstad, J.J. Derksen, and H.E.A. van den Akker. Piv measurements and lattice boltzmann simulations on a single sphere settling under gravity. *Physics of Fluids*, 14(11): 4012–4025, 2002.

[33] Unknown. Open source lattice boltzmann code.