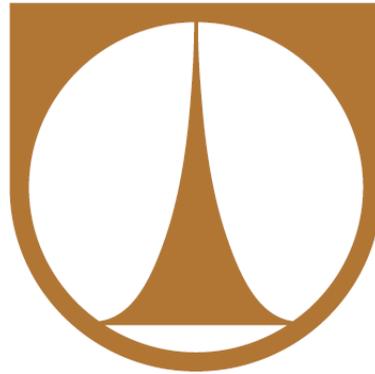


**TECHNICAL UNIVERSITY OF LIBEREC**

**FACULTY OF TEXTILE ENGINEERING**



**Ing. LARYSA OCHERETNA**

**THE LATTICE GAS CELLULAR AUTOMATA APPROACH FOR  
FLUID FLOWS IN POROUS MEDIA**

**DOCTORAL THESIS**

**2012**



**TECHNICAL UNIVERSITY OF LIBEREC**

**FACULTY OF TEXTILE ENGINEERING**

Department of Nonwovens and Nanofibrous Materials

Doctoral thesis

**THE LATTICE GAS CELLULAR AUTOMATA APPROACH FOR FLUID  
FLOWS IN POROUS MEDIA**

Ing. Larysa Ocheretna

Advisor: Prof. RNDr. David Lukáš, CSc., FT, TU of Liberec

**Thesis contains:**

Number of Pages: 96

Number of Figures: 50

Number of Tables: 4

Number of Appendixes: 19

## **Declaration on word of honour**

I, Larysa Ocheretna, declare that this thesis has been elaborated independently with the support of mentioned literary sources.

In Liberec 18<sup>th</sup> December 2012

Larysa Ocheretna

## **ACKNOWLEDGE**

Foremost, I would like to sincerely thank my supervisor, Prof. RNDr. David Lukáš, CSc. for his encouragement, guidance and support to me throughout my PhD study. His patience on the one side and motivation on the other, enthusiasm and, of course, immense knowledge of the studied problem helped me a lot in all stages of the study, research and writing up the PhD thesis.

I am grateful to my former colleagues from the Department of Nonwovens (Technical university of Liberec) for a number of projects that I participated together with them during my PhD study. Furthermore, special thank belongs to Ing. Eva Košťáková, Ph.D. and deceased Martin Hamouz for their friendly support, acquaintance with Czech culture and wonderful time, that I enjoyed together with them.

I would also like to thank my colleagues from the Department of Textile Evaluation (Technical university of Liberec) for their support during the final stage of my PhD study.

I thank Stephen Wolfram for his valuable lectures and advices in relation to the Cellular Automata provided during Wolfram Science Summer School.

I am grateful to Victoria Vlasenko (Kyjiv National University of Technology and Designs) for her faith in me, recommendation for doctoral studies in Technical university of Liberec and nearly parental care through all period of my study.

My deepest gratitude goes to my family. My parents, my brother, and my boyfriend were sources of my strength. Without their love, understanding and encouragement it would have been impossible for me to complete the work.

Finally I would like to thank all who have supported me during my PhD study.

## **ABSTRACT**

The thesis is focused on the modelling of fluid flow in porous media. The aim of the work was to develop an appropriate model for simulation of fluid transport regardless of the flow regime.

The model, developed in the frames of the work, is based on Lattice Gas Cellular Automata. The model is non-deterministic and fully discrete. It is presented by means of algorithm created in a C++ programming language. The algorithm allows computer simulation of the fluid flow through different porous structures, including nanofibre materials, where the pore size is on the order of free path of molecules and flow thus loses its continuous properties.

The model is verified for two phenomena as the Brownian motion and Poiseuille flow are. The presented model is used to the study of fluid flow inside assembled filters with different density of porous media. Simulation results proved the hypothesis regarding to the reorganization of the flow inside the filter and its orientation perpendicularly to the pleat surface.

## **ANOTACE**

Předložená disertační práce je zaměřena na modelování proudění tekutiny porézním prostředím. Cílem práce bylo vytvoření vhodného modelu pro simulaci transportu tekutiny nezávisle na režimu jejího proudění.

Předložený model vychází z podstaty buněčných automatů a využívá rysy mřížového plynu. Model je nedeterministický a plně diskrétní. Pomocí programu vytvořeného v C++ programovacím prostředí umožňuje počítačovou simulaci a studium proudění tekutiny různými porézními strukturami, včetně nanomateriálů, kde velikosti pórů řádově se blíží délce volné dráhy molekuly a proudění tak ztrácí své kontinuální vlastnosti.

Funkce modelu jsou ověřeny pomocí dvou testů, tj. simulací Brownova pohybu a Poiseuillova proudění. Předložený model je použit na studium proudění tekutiny skládanými filtry s různou hustotou porézního prostředí. Výsledky simulací prokazují hypotézu týkající se orientace proudění kolmo k povrchu skladů filtru.

# TABLE OF CONTENTS

LIST OF PICTURES .....	8
LIST OF TABLES.....	10
LIST OF SYMBOLS AND ABBREVIATIONS.....	11
INTRODUCTION.....	14
<b>1. BASIC PRINCIPLES OF MODELLING AND COMPUTER SIMULATION .....</b>	<b>16</b>
1.1. ORIGINS AND DEVELOPMENT OF MODELLING AND COMPUTER SIMULATION .....	16
1.2. MODEL: THE DEFINITION AND CLASSIFICATION .....	17
1.3. SIMULATION STUDY AND COMPUTER SIMULATION: DEFINITIONS, STAGES, BENEFITS AND DANGERS OF THEIR IMPLEMENTATION .....	21
1.4. MODELLING AND SIMULATION IN THE TEXTILE INDUSTRY.....	23
1.4.1. <i>Navier-Stokes equation</i> .....	26
1.4.2. <i>Boltzmann equation</i> .....	29
<b>2. MODELLING WITH CELLULAR AUTOMATA AND LATTICE GAS CELLULAR AUTOMATA.....</b>	<b>32</b>
2.1. HISTORICAL OVERVIEW: CELLULAR AUTOMATA AND LATTICE GAS AUTOMATA .....	32
2.2. SPECIFICATION OF FINITE AUTOMATA, CELLULAR AUTOMATA AND LATTICE GAS CELLULAR AUTOMATA .....	35
2.2.1. <i>Finite automata</i> .....	35
2.2.2. <i>Cellular automata</i> .....	38
2.2.3. <i>Lattice gas cellular automata as a special case of cellular automata</i> .....	40
2.3. PRINCIPLES OF LATTICE GAS CELLULAR AUTOMATA .....	41
2.3.1. <i>Discretization of space – basic methods. Grid generation</i> .....	43
2.3.2. <i>Discretization of space in LGCA model</i> .....	45
2.3.2.1. Geometry of square and hexagonal lattices.....	48
2.3.2.2. Neighbourhoods in the square and hexagonal lattice .....	50
2.3.2.3. Comparison of square and triangular lattice with hexagonal symmetry.....	54
2.4. LATTICE GAS CELLULAR AUTOMATA – PRINCIPLES OF THE MODEL .....	54
2.4.1. <i>Collision phase</i> .....	56
2.4.1.1. Collision rules of the FHP-1 and FHP-2 LGCA models.....	58
2.4.2. <i>Propagation phase in LGCA models</i> .....	61
<b>3. BASIC ALGORITHM BASED ON THE FHP-1 LATTICE GAS CELLULAR AUTOMATA.....</b>	<b>63</b>
3.1. CODE FRAGMENT 1 – HEADER FILES AND INITIALIZATION OF THE SIMULATION DOMAIN .....	63
3.2. CODE FRAGMENT 2 – GRAPHIC OUTPUT SETTING.....	65
3.3. CODE FRAGMENT 3 – CREATION OF THE SIMULATION DOMAIN AND INITIAL STATE OF THE SIMULATED SYSTEM .....	65
3.4. CODE FRAGMENT 4 – OCCUPATION OF CHANNELS BY FLUID PARTICLES .....	66
3.4.1. <i>Geometry of the lattice</i> .....	66
3.4.2. <i>Occupation of channels by fluid particles</i> .....	68
3.5. CODE FRAGMENT 5 – GRAPHICAL OUTPUTS OF THE INITIAL SYSTEM CONFIGURATION .....	69
3.6. CODE FRAGMENT 6 – THE MAIN CYCLE OF THE ALGORITHM.....	69
3.6.1. <i>Code fragment 6-A – Collision phase</i> .....	72
3.6.2. <i>Code fragment 6-B – Propagation phase</i> .....	73
3.7. CODE FRAGMENT 7 – RECORDING OF THE NEW SYSTEM’S STATE.....	74
3.8. CODE FRAGMENT 8 – DATA ARRAYS RESETTING .....	75
3.9. CODE FRAGMENT 9 – PRINTOUT MACRO .....	75
3.10. CODE FRAGMENT 10 – FINAL OPERATIONS .....	76
<b>4. VARIFICATION OF FHP-1 LGCA ALGORITHM FOR BROWNIAN MOTION.....</b>	<b>77</b>

4.1.	THEORETICAL ASSUMPTION .....	77
4.2.	FHP-1 LATTICE GAS CELLULAR AUTOMATA ALGORITHM FOR BROWNIAN MOTION SIMULATION .....	78
4.2.1.	<i>Code fragment 1 – Header files and initialization of the simulation box.....</i>	78
4.2.2.	<i>Code fragment 4 – Occupation of channels by fluid particles.....</i>	79
4.2.3.	<i>Code fragment 5-A – Data outputs .....</i>	79
4.2.4.	<i>Code fragment 6 – The main cycle of the algorithm .....</i>	80
4.2.4.1.	Code fragment 6-A – Collision phase.....	80
4.2.4.2.	Code fragment 6-B – Propagation phase .....	81
4.2.5.	<i>Code fragment 9 – Printout macro.....</i>	82
4.3.	SIMULATION SETUP .....	82
4.4.	RESULTS AND DISCUSSION .....	85
<b>5.</b>	<b>VARIFICATION OF THE FHP-1 LGCA ALGORITHM FOR POISEUILLE FLOW .....</b>	<b>87</b>
5.1.	THEORETICAL ASSUMPTION .....	87
5.2.	FHP-1 LATTICE GAS CELLULAR AUTOMATA ALGORITHM FOR POISEUILLE FLOW SIMULATION .....	88
5.2.1.	<i>Code fragment 1 – Header files and initialization of the simulation box.....</i>	89
5.2.2.	<i>Code fragment 3 – Creation of the simulation domain .....</i>	89
5.2.3.	<i>Code fragment 5-A – Data outputs .....</i>	90
5.2.4.	<i>Code fragment 6 – The main cycle of the algorithm .....</i>	90
5.2.4.1.	Code fragment 6-B – Pressure gradient .....	92
5.2.4.2.	Code fragment 6-C – Propagation phase .....	93
5.2.5.	<i>Code fragment 9 – Printout macro.....</i>	94
5.3.	SIMULATION SETUP .....	94
5.4.	RESULTS AND DISCUSSION .....	96
<b>6.</b>	<b>COMPUTER SIMULATION OF THE TWO-DIMENSIONAL FLUID FLOW THROUGH POROUS STRUCTURES</b>	
	<b>101</b>	
6.1.	THEORETICAL ASSUMPTION .....	101
6.2.	FHP-1 LATTICE GAS CELLULAR AUTOMATA ALGORITHM FOR FLUID FLOW THROUGH POROUS MEDIUM SIMULATION	102
6.2.1.	<i>Code fragment 1 – Header files and initialization of the simulation domain.....</i>	102
6.2.2.	<i>Code fragment 3 – Creation of the simulation domain .....</i>	103
6.2.3.	<i>Code fragment 6 – The main cycle of the algorithm .....</i>	104
6.2.4.	<i>Code fragment 9-B – Distribution of velocity vectors of moving particles.....</i>	104
6.3.	SIMULATION SETUP .....	104
6.4.	RESULTS AND DISCUSSION .....	106
	<b>CONCLUSIONS.....</b>	<b>111</b>
	FUTURE WORK .....	113
	<b>REFERENCES .....</b>	<b>114</b>
	<b>PUBLICATIONS OF AUTHOR.....</b>	<b>120</b>
	<b>LIST OF APPENDIXES.....</b>	<b>122</b>

# LIST OF FIGURES

<b>Figure 1:</b> Types of dynamic models: <b>a</b> – continuous-time model, <b>b</b> – discrete-time model.....	19
<b>Figure 2:</b> Scheme of model classification .....	21
<b>Figure 3:</b> Stages of simulation study .....	22
<b>Figure 4:</b> Different regimes of fluid flow and methods for their description depending on Knudsen number ....	25
<b>Figure 5:</b> The acceleration of fluid unit volume.....	27
<b>Figure 6:</b> Set of patterns obtained in game of “Life” for various time evolution steps $t$ .....	34
<b>Figure 7:</b> Finite automaton represented using classical methods: state-transition table, state tree and state diagram [17] .....	37
<b>Figure 8:</b> Basic principle of a finite automaton operation.....	37
<b>Figure 9:</b> Graphical explanation of the cell having the position $\mathbf{r}$ and its neighbour cells located in a regular square lattice.....	39
<b>Figure 10:</b> Graphical interpretation of two-dimensional cellular automaton: <b>a</b> – general appearance of a regular lattice, <b>b</b> – detailed configuration of neighbourhood cells of reference cell, <b>c</b> – application of a transition function and updating the state of the cell at time $t+1$ [17] .....	40
<b>Figure 11:</b> Cell of the cellular automaton as a individual automaton: states of the neighbour cells are inputs, the new state as an output of the automaton [17] .....	40
<b>Figure 12:</b> Two-dimensional Bravais lattices: <b>a</b> - square, <b>b</b> - rectangular, <b>c</b> - oblique, <b>d</b> - centered rectangular, <b>e</b> – hexagonal [17].....	47
<b>Figure 13:</b> Types of the square lattice: upright ( <b>a</b> ) and diagonal one ( <b>b</b> ) [64] .....	49
<b>Figure 14:</b> Geometries of 2D hexagonal lattices: <b>1</b> – hexagonal lattice with horizontal ( <b>a</b> , <b>b</b> ) or vertical ( <b>c</b> ) rows; <b>2</b> – hexagonal honeycomb lattice with vertical ( <b>a</b> , <b>b</b> ) or horizontal ( <b>c</b> ) rows .....	49
<b>Figure 15:</b> Neighbourhood templates for a regular square lattice: von-Neumann neighbourhood ( <b>a</b> ), Moore neighbourhood ( <b>b</b> ) and Margolus neighbourhood ( <b>c</b> ) .....	51
<b>Figure 16:</b> The hexagonal neighbourhood .....	52
<b>Figure 17:</b> Mersereau's scheme for obtaining the hexagonal lattice ( <b>b</b> ) from the square one ( <b>a</b> ) .....	52
<b>Figure 18:</b> Staunton's method for obtaining hexagonal lattice ( <b>b</b> ) from the square one ( <b>a</b> ).....	53
<b>Figure 19:</b> Adaptation of the hexagonal neighbourhood to the square lattice: the ordering of the neighbour nodes in all odd ( <b>a</b> ) and even ( <b>b</b> ) rows.....	53
<b>Figure 20:</b> Representation of the LGCA model underlaid by the hexagonal Bravais lattice: <b>1</b> – the node, i.e. the individual automaton, <b>2</b> – the channel, <b>3</b> – the moving particle, <b>4</b> – the direction of moving [17].....	56
<b>Figure 21:</b> Typical two- and three-particle collisions in the FHP-1 LGCA model [17].....	58
<b>Figure 22:</b> Effective collisions in the FHP-1 LGCA model [17].....	59
<b>Figure 23:</b> Two- and three-particle collisions in FHP-2 LGCA model [17].....	61
<b>Figure 24:</b> The principle of periodic boundary conditions for two-dimensional square LGCA (HPP model [17]) ..	62
<b>Figure 25:</b> Various reflective boundary conditions: <b>A</b> - bounce-back reflection, <b>B</b> - specular reflection, <b>C</b> - diffusive reflection [17].....	62
<b>Figure 26:</b> Various examples of node's occupation: <b>A</b> – an empty node, <b>B</b> – node occupied by a solid particle, <b>C</b> – the node occupied by fluid particles .....	66
<b>Figure 27:</b> The hexagonal lattice as the equivalent square lattice with an additional diagonal connection ( <b>a</b> ) and the regular hexagonal neighbourhood in odd and even rows of the lattice ( <b>b</b> ) .....	67
<b>Figure 28:</b> The ordering of the channels $i_1 \dots i_6$ and the determination of the neighbour nodes position in all odd ( <b>a</b> ) and even ( <b>b</b> ) rows of the lattice.....	68
<b>Figure 29:</b> The initial configurations of the system according to the value of the parameter $p_{co}$ : $p_{co} = 2$ ( <b>a</b> ); $p_{co} = 20$ ( <b>b</b> ); $p_{co} = 200$ ( <b>c</b> ) and $p_{co} = 2000$ ( <b>d</b> ). .....	70
<b>Figure 30:</b> The flowchart representing the main cycle of the developed FHP-1 LGCA algorithm.....	71
<b>Figure 31:</b> Monitoring of the simulated system. The state after application of 110 cycles .....	75
<b>Figure 32:</b> Simulation of the Brownian motion presented on the reduced simulation domain $30 \text{ l. u.} \times 30 \text{ l. u.}$ : <b>a</b> – the simulation domain bounded by solid walls (red lines), moving particles (blue squares), initial	

position of the Brownian particle (grey square) and its final position (yellow square) after 20 time steps, black squares present empty lattice nodes; <b>b</b> – Brownian random motion during 20 time steps obtained by the developed model based on the FHP-1 LGCA model.....	83
<b>Figure 33:</b> Displacement $R$ of the Brownian particle .....	84
<b>Figure 34:</b> Paths of the Brownian particle after 4000 time steps: <b>a</b> – the straight type of paths (simulation experiment 1, data output brown04.cpp); <b>b</b> – the “bonsai tree” shape of the path (simulation experiment 1, data output is brown11.cpp) .....	86
<b>Figure 35:</b> The main square displacement of the Brownian particle as a function of time, for $\rho=1,5$ particles/node and 3 particles/node.....	86
<b>Figure 36:</b> The geometry of two-dimensional channel for Poiseuille flow simulation: <b>1</b> – periodic boundary conditions, <b>2</b> – the imaginary ventilator, <b>L</b> is the length and <b>d</b> is the width of the channel.....	90
<b>Figure 37:</b> The flowchart representing the main cycle of the algorithm developed for a simulation of the Poiseuille flow .....	91
<b>Figure 38:</b> An example of the forced reorganization of channel occupation. Propagation of moving particle from the channel $i1$ to $i4$ .....	92
<b>Figure 39:</b> Propagation of moving particles at the left (a) and at the right (b) boundaries of the channel. Periodic boundary conditions are applied.....	93
<b>Figure 40:</b> The flow rate as a function of time for $L = 550$ l.u., $d = 1003/2$ l.u., $f_x = 0,3$ . The time period of the simulation measured in time units (t.u.) is given at the axis OX. Steady state of the flow is achieved after about 5000 t.u.....	97
<b>Figure 41:</b> The velocity profile of the flow. Values of the x component of flow velocity averaged over the whole channel length (i.e. 550 l.u.) are at the axis OY. The vertical distance from the bottom wall of the channel named here as a “axis OY” and it is presented at the axis OX of the graph .....	97
<b>Figure 42:</b> Predicted and simulated volumetric flow rate as a function of channel width for a pressure gradient created using $f_x = 0,4$ and the range of the channel width $d=25\div 100$ l.u. ....	98
<b>Figure 43:</b> Predicted and simulated volumetric flow rate as a function of channel width for a pressure gradient created using $f_x = 0,4$ and the range of the channel width $d=25\div 200$ l.u. ....	99
<b>Figure 44:</b> Theoretical flow pattern through pleats at assembled filter .....	102
<b>Figure 45:</b> The geometry of two-dimensional channel for fluid flow through porous medium simulation: <b>L</b> is the length and <b>d</b> is the width of the channel, $\alpha$ is an inclination angle of the porous medium, <b>i</b> is a one half of the porous medium thickness, <b>1</b> – periodic boundary conditions, <b>2</b> – the imaginary ventilator. The vertical dot line presents the vertical axis of the channel.....	103
<b>Figure 46:</b> Random structures of porous media generated in the computer simulation experiment. Porosity ranging from 0,7 to 0,95 .....	106
<b>Figure 47:</b> Fluid flow rate as a function of porosity and inclination of porous medium for pressure gradient created using $f_x = 0,6$ .....	107
<b>Figure 48:</b> Pressure gradient created using $f_x = 0,6$ as a function of inclination angle $\alpha$ indicates the orientation of the porous medium in a channel.....	108
<b>Figure 49:</b> Fluid velocity directions inside the decline porous material with random structure for porosity 0,95 and $\alpha=15^\circ, 35^\circ$ and $55^\circ$ . Region BP corresponds to “blind pores” of the porous medium.....	109
<b>Figure 50:</b> Fluid velocity directions inside the channel and decline porous material with random structure for porosity 0,7 and $\alpha= 35^\circ$ .....	110

## LIST OF TABLES

<b>Table 1:</b> Characterization of different types of grids.....	45
<b>Table 2:</b> The list of Brownian motion computer simulations and their setups.....	83
<b>Table 3:</b> The list of Poiseuille flow computer simulations and their setup.....	94
<b>Table 4:</b> The list of fluid flow through porous medium computer simulations and their setups.....	104

# LIST OF SYMBOLS AND ABBREVIATIONS

## Symbols

Symbol	Meaning
$A_i$	parameter that affect the behaviour of the system
$F_i$	function that define the system
$P_1$	starting position of the fluid unit
$P_2$	final position of the fluid unit
$S_0$	initial state of FA/CA
$S_i$	current state of FA/CA
$X_i$	physical or other variable
$a_i$	input signal of FA
$a_i$	integer coefficient
$d_p$	particle diameter
$f_x$	change of the $x$ component of the particle momentum
${}^n n_i$	“new” local particle number
${}^n p_i$	“new” local momentum
$n_i$	local particle number
$p_i$	local momentum
$v_i$	local velocity
$v_{ia}$	local component of velocity vector
$v_x$	$x$ component of velocity vector
$v_y$	$y$ component of velocity vector
$v_z$	$z$ component of velocity vector
$\mathbf{R}^n$	vector space
$\mathbf{e}_i$	unit vector
$\mathbf{f}_{visc}$	viscous force
$\lambda_i$	position vector of the neighbour node
$\nabla$	divergence
$k_B$	Boltzmann’s constant
$p$	pressure
$t$	time
$\alpha$	Inclination angle of a porous medium
$D$	dimension
$D$	diffusion coefficient
$F$	set of final states of FA
$G$	pressure gradient
$Kn$	Knudsen number
$L$	length
$N$	number of lattice nodes
$S$	state of FA/CA

$T$	temperature
$b$	coordination number
$f$	distribution function
$i$	channel label
$k$	permeability of the medium
$l$	distance between neighbour nodes
$m$	mass
$n$	number of time steps
$q$	flow rate
$t$	temperature
$x$	coordinate, corresponds to the axis OX
$y$	coordinate, corresponds to the axis OY
$z$	coordinate, corresponds to the axis OZ
$R$	vector distance of the Brownian particle
$a$	acceleration vector
$f$	force per unit volume
$r$	position vector
$v$	velocity vector
$\delta$	state-transition function, also an update rule
$\lambda$	mean free path of molecule
$\mu$	dynamic viscosity
$\rho$	density
$\tau$	relaxation time
$\varphi$	potential per unit mass
$\Omega$	vector field

## Subscripts

Subscript	Meaning
$l. u.$	lattice unit
$m. u.$	mass unit
$t. s.$	time step
$t. u.$	time unit

## Abbreviations

Abbreviation	Meaning
C++	programming language
CA	cellular automata
CSL	control and simulation language
DSMC	direct simulation Monte Carlo
E	east node

FA	finite automata
FDM	finite differences method
FEM	finite elements method
FHP	lattice gas cellular automata at the hexagonal lattice (called after Frisch, Hasslacher, Pomeau)
FVM	finite volume method
GPSS	general purpose simulation system
HPP	lattice gas cellular automata at the square lattice (called after Hardy, de Pazzis, Pomeau)
LBM	lattice Boltzmann model
LGCA	lattice gas cellular automata
LL	lower-left node
LR	lower-right node
MD	molecular dynamic
N	north node
NE	north-east node
NW	north-west node
S	south node
SE	south-east node
SIMSCRIPT	simulation programming language
SIMULA	programming language
SPH	smooth-particle hydrodynamics
SW	south-west node
UL	upper-left node
UR	upper-right node
W	west node

## INTRODUCTION

Fluid flow and especially fluid flow in porous media is a subject of wide interest for a long time. From the beginning of the 19<sup>th</sup> century thanks to *Claude-Louis Navier* and *George Gabriel Stokes* fluid motion has got a solution in a form of Navier-Stokes differential equations. These equations have arisen, when macroscopic nature of the fluid was only known. A continuum fluid flow was a subject of study at that time. The validity of Navier-Stokes approach remained undeniable until today. Navier-Stokes equations became a core of the most part of modern software designated for fluid flow modelling, including fluid flow in porous structures.

If we evaluate current scientific trends in global, and textile engineering especially, nanomaterials became the subject of the study in all branches of science and research. Revolutionary material of the 3<sup>rd</sup> Millenium, nanofibre and nanoparticle materials, and development of the textile materials with difficult internal structures (i.e. multilayer textile structures) requires a deeper reassessment of theoretical techniques and methods, used for a fluid flow description so far.

Before any the newly developed textile becomes the subject of business, a number of experimental work is could to be done for a determination of its properties. Not all properties can be evaluated using available experimental methods and techniques. Therefore, the demand for modelling and computer simulations is increasing. The more the characteristic dimension of the object under investigation decreases, the exploration of its properties becomes more complicated and expensive. Moreover, modelling and simulations are often used in order to: (i) obtain critical values of particular parameters of a object or a phenomenon; (ii) visualize the time evolution of the phenomenon; (iii) verify empirically obtained results.

Since the fully discrete model of hydrodynamics based on cellular automata conception was developed and verified for fluid flow, more and more researchers become to use this approach in modelling and simulation. Lattice Gas Cellular Automata appears to be very simple at first glance. Nevertheless it provides the more number of options for modelling of fluid flow in contrast to Navier-Stokes equations. Because of its discrete nature it doesn't have limitations in continuity of the flow. It is valid in all regimes of flow – from the molecular flow to the continuum one.

In this dissertation, several contributions to the study of the fluid flow mechanism by means of Lattice Gas Cellular Automata method are presented. The motive why Lattice Gas Cellular Automata were chosen for fluid flow modelling and simulation is presented in the Chapter 1. First, the basic principles of modelling and computer simulation are here described. Then the current state of the modelling and simulation in textile industry and especially methods for fluid flow modelling are discussed. The substance of the Navier-Stokes and the Lattice Boltzmann approaches are presented in the second part of the Chapter 1. Lattice Gas Cellular

Automata model, based on the Lattice Boltzmann approach is described from its origin in the Chapter 2. A great attention is paid here to the principles of the space discretization and to the description of the different lattice properties, which are very important during the creation of an Lattice Gas Cellular Automata model and its application.

The detailed description of the Lattice Gas Cellular Automata algorithm developed for a fluid flow simulation is presented in the Chapter 3. This algorithm was verified for two phenomena as the Brownian motion and the Poiseuille flow are. The basic algorithm was adjusted for these benchmark tests. Related algorithms and results obtained from the computer simulations are subsequently presented in Chapters 4 and 5. Application of the developed Lattice Gas Cellular Automata for fluid flow in a porous medium simulation is presented in the Chapter 6 of the thesis. Computer simulation based on the developed Lattice Gas Cellular Automata algorithm verifies here the particular hypothesis related to the curious behaviour of the fluid flow through assembled filters. General summary of the work included visions for the future are presented in the conclusions of the thesis.

## BASIC PRINCIPLES OF MODELLING AND COMPUTER SIMULATION

*“How can it be that mathematics, being after all a product of human thought independent of experience, is so admirably adapted to the objects of reality?”*

*Albert Einstein*

Many researchers, which deal with modelling, claim that current research in the natural or social science can no longer be imagined without simulations, especially computer ones. What was the way of modelling and computer simulation developing, which models are known at present time, what stages are the part of simulation study, which benefits and dangers of simulation study and partly computer simulation entails, is described in this capture.

### 1.1. Origins and development of modelling and computer simulation

Without any doubt, first models were already designed in ancient time. It is known, that ancient Egyptians created all sorts of models. It is possible, that first physical models come from Egypt – models of their tools, vessels, weapons or boats and other objects are founded in a big amount in their tombs and serve to the study of this ancient culture now. In ancient time those models were used to assure that a human be taken care of during the afterlife.

In fact, modelling as a theoretical activity began to be dominating at first in the field of physics in the end of 19<sup>th</sup> century. For example, *J.C. Maxwell* to derive the equation of electromagnetism used analogical hydrodynamic models. *Lord Kelvin* (originally *William Thomson*) mentioned that he couldn't understand a phenomenon until he had built a mechanical model of the system under consideration [1].

Simultaneously, development of modelling was linked with the invention of computer technology and its implementation into the technical sciences. The concept of a first computing machine was intimated in a series of drawings of reduction *Charles Babbage* between 1834 and 1857. His so-called “Analytical Engine” was designed to perform calculations automatically with a possibility of simple programming [2]. But first computer simulation models appear during World War II. On the one side analog computer was well known in a world of science, on the other side the development of the first nuclear weapon was initiated within the frame of Manhattan Project and the two mathematicians *Jon von Neumann* and *Stanislaw Ulam* using Monte Carlo approach tried to understand the puzzling problem of behaviour of neutrons at that time. The real experimentations were too costly and the problem was too complicated for analysis [1, 3]. In the late 1940s and early 1950s,

both analog and digital<sup>1</sup> computers started to appear in a number of organizations. In the 1950s, the computers were used for census data recording, defence systems, accounting and some scientific calculation. The development of programming languages was felt, first of them were rising during the 1960s:

- SIMSCRIPT (*Markowitz H., Hausner B., Karr H.*) – simulation programming language developed in 1962 for the U.S Air Force [4];
- CSL – the Control and Simulation Language (*Buxton J., Laski J.*) designed for use in the field of complex logical problems. The first application has been in the field of Monte Carlo simulation [5];
- SIMULA (*Dahl O., Nygaard K.*) – originally it was designed and implemented as a language for discrete event simulation, than it was reimplemented as a general purpose programming language. Simula-type objects were later implemented in C++, Java and C# programming languages [6].

In the 1970s, simulation was a topic that was taught to industrial engineers but rarely applied. Long time spent at the computer terminal and endless runs to find a bug in a language was what “simulation” meant at that time. The popularity of simulation as a powerful tool rapidly increased with the number of conferences and seminars devoted to this problem. According to *Reitman* [7] first of them were: Conference on Simulation Language (1964), Conference on Application of Simulation using the General Purpose Simulation System (GPSS) (1967), Application of Simulation (1968) and Winter Simulation Conference (1971) that is also popular at the present time. The number of sessions held to computer simulation within the frame of conferences was quintuple at the beginning of 1980s compare with the end of 1960s. In the 1980s, the offer of computerized systems was very limited and too expensive. The number of companies using computer simulations was still small. The first simulation language specifically designed for modelling manufacturing systems and the discrete event simulation model was developed in 1984. In the middle of 1990s the power of simulation as a tool became evident and popular [8]. A big amount of simulation packages represented both by simulation languages and application-oriented simulators is in offer at present time [9], and modelling in itself became more and more popular in technology.

### 1.2. Model: the definition and classification

Models are considered to be one of the basic instruments of modern science. Formally, a model is defined as a formalized interpretation, which uses symbols instead meanings,

---

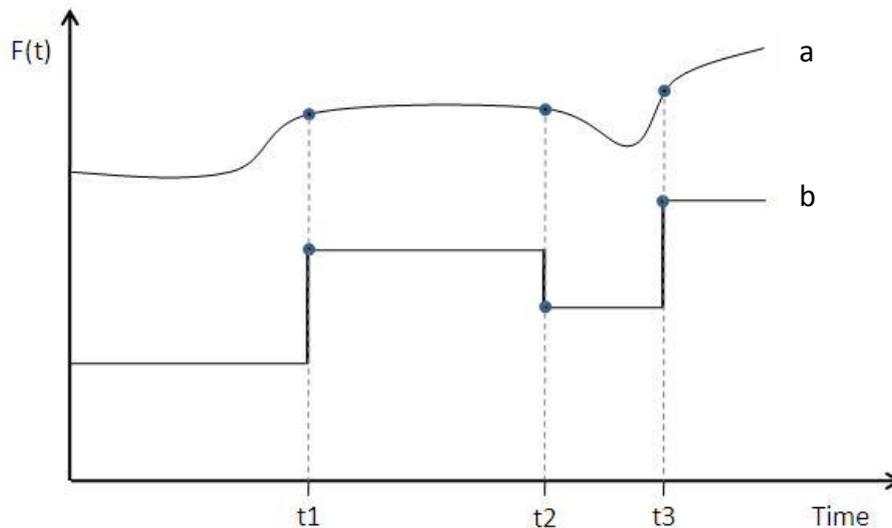
<sup>1</sup> In electronics and computer science *analog computer* is defined as a mechanical, electrical, or electronic computer that performs arithmetical operations by using some variable physical quantity, such as mechanical movement or voltage, to represent numbers. *Digital computer* is an electronic computer in with the input is discrete rather than continuous, consisting of combinations of numbers, letters and other characters written in an appropriate programming language and represented internally in binary number system (116).

substitutes truth-values with the sentences of a formal language. Depending on using and representation several kinds of models are mentioned in literature [10]:

- *Mental model* – describes person's behaviour in different situations. In other words, it is an explanation of person's thought process according to surrounding world, and relation to its parts.
- *Verbal model* – consists of intuitive concepts, often used for mathematical models interpretation. In contrast to mathematical model, verbal model doesn't have exact and logical internal structure, consequently the verbal model is considered to be slightly ambiguous and inaccurate.
- *Physical model* – this term is often used in literature for the computer simulation model of the certain physical system signification. In fact, it is a small physical object with the same shape and appearance as the real object to be studied. Physical models mimic some properties of real systems.
- *Mathematical model* – gives description of real system or phenomenon, where the relationships between variables of the system are expressed in mathematical form using mathematical language. So, a great number of laws of nature are mathematical models.

The kinds of models that will be dealt with in this work are mathematical models represented by means of computer simulation algorithms. The detail classification of mathematical models is given below.

There are *static* and *dynamic* mathematical models with respect to model behaviour in time. Static model describes the system in steady state, where the physical characteristics have constant values. Dynamic model includes time. The time development of a system (the change of its outputs in dependence on the same inputs) is the subject of study here. The changing of values of any parameter in time is often an output of the dynamic model. There are two main classes of dynamic models depending on how the function changes its character in time: *continuous-time* and *discrete-time* models (see *Figure 1*). Continuous-time models evolve their variable values continuously over time, while discrete-time models change their variable values at discrete points in time only. [10]



**Figure 1:** Types of dynamic models: **a** – continuous-time model, **b** – discrete-time model

Mathematical models could be denoted also as a *qualitative* or *quantitative*. Mainly, qualitative analysis is used in social studies and is thought to be subjective and non-statistical. Qualitative models involve an in-depth understanding of system behaviour and the reason of such behaviour. Unlike quantitative models, which rely exclusively on the analysis of numerical or quantifiable data and their outputs are represented by means of mathematical formulas or graphs. In qualitative models (or analysis) the images, sound, video and text is often working with.

The most part of phenomena in nature are preceded as non-deterministic processes. Mathematical non-deterministic models are called *stochastic* or *probability-based* models. The stochastic process is defined as a one whose behaviour is non-deterministic and the next state is determined both by process's predictable actions and by random element. In other words, the stochastic model is a mathematical representation of random phenomena, which is defined by sample space, events within the space and probabilities associated with each event [11]. The counterpart of the stochastic is a *deterministic* model, which is specified by a set of known relationships among states and events without any random variation. If the stochastic model is run several times, it will not give identical results, while in deterministic model the given input will always produce the same output. The most common types of stochastic modelling tasks are:

- Markov chains and processes describing the evolution of dynamic processes;
- Economic models of supply and demand;
- Survival models (in insurance and health);
- Game models that have application in strategic decision making.

It is interesting, that dynamic processes can be modelled using the both deterministic and stochastic (non-deterministic) ways. According to [12] dynamic processes are usually described by means of a set of first order differential equations:

$$\frac{dX_i}{dt} = F_i(X_1, X_2, \dots, X_n, A_1, A_2, \dots, A_s), \quad i = 1, 2, \dots, n. \quad (1)$$

where  $X_i$  are physical or other variables;  $t$  is time;  $F_i, i = 1, 2, \dots, n$  are functions that define the system;  $A_1, A_2, \dots, A_s$  are parameters that partly affect the behaviour of the dynamic system (different constants and values of external parameters, etc.). Depending on the values of parameters  $A_1, A_2, \dots, A_s$  the behaviour of the system can be regular and orderly or irregular and disordered. But the core of a random non-deterministic behaviour of the system is not the large number of degrees of freedom or uncontrollable external factors, but mainly non-linear internal dynamics, leading to instability and chaotic behaviour. Looking back at the *Equation 1*, when the function  $F_i$  is non-linear (for example,  $F_i = X_1^3 + \cos X_1^2$ ), then  $\frac{dX_i}{dt}$  becomes non-linear also. Due to non-linearity the system loses memory – ie. a record of its initial conditions. Then the statistical description (stochastic model) is not only possible but actually the only effective and suitable one.

According to [13], all above-mentioned models represent phenomena and/or data in general.

Representational models of phenomena are:

- *Scale models* – are basically miniaturized or enlarged copies of their real systems; they provide faithful copy of the shape, but not the material.
- *Idealized models* – are simplified models of complicated systems. Two general kinds of idealized models are under consideration: models based on a so-called Aristotelian and/or Galilean idealizations. Aristotelian idealization is equal to “stripping away”, in other words all properties of the real system that we believe aren’t significant to our model are being disregard. Galilean idealization involves deliberate distortion of the model towards real system. Aristotelian and Galilean idealization are often come together in models.
- *Analogical models* – represent the target systems or phenomena by another more understandable system if there are certain relevant similarities between them.
- *Phenomenological models* – those models are considered to be independent of theories, they result from different empirical observation of the target system or phenomena.

Representational models of data are idealized versions of the data gained from immediate observation. Mainly mathematical models are ranged between them. The full overview of models mentioned in this chapter and their sections is presented in the *Figure 2*.

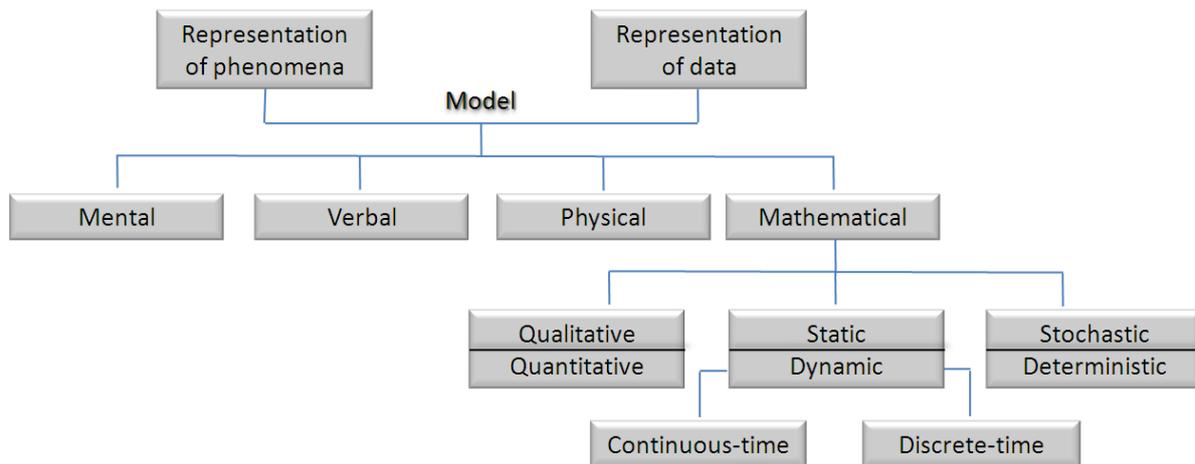


Figure 2: Scheme of model classification

The process of producing a model is considered to be *modelling* [9]. More about modelling and computer simulation especially is presented in the Chapter 1.3.

### 1.3. Simulation study and computer simulation: definitions, stages, benefits and dangers of their implementation

*Modelling* is understood as a process of model generation. *Simulation* is an imitation of the real process or phenomenon over the time and it includes several stages (see Figure 3). The term “simulation” comes from Latin “simulare” and means “to pretend” [10]. “Simulation” often occurs in connection with dynamic mathematical models – as an experiment performed on a model. The aim of simulation is to solve the equation of motion of such a model and herewith to represent the time-evolution of the target<sup>2</sup> system [13]. But generally simulation is defined in literature as a tool to evaluate the performance of a system, existing or proposed, under different configurations of interest and over the time.

Usually, simulation is used when an existing system should be altered or a new system built [9]. System here is an object or collection of objects whose properties we want to study. Two reasons for system study are mentioned in literature [10]:

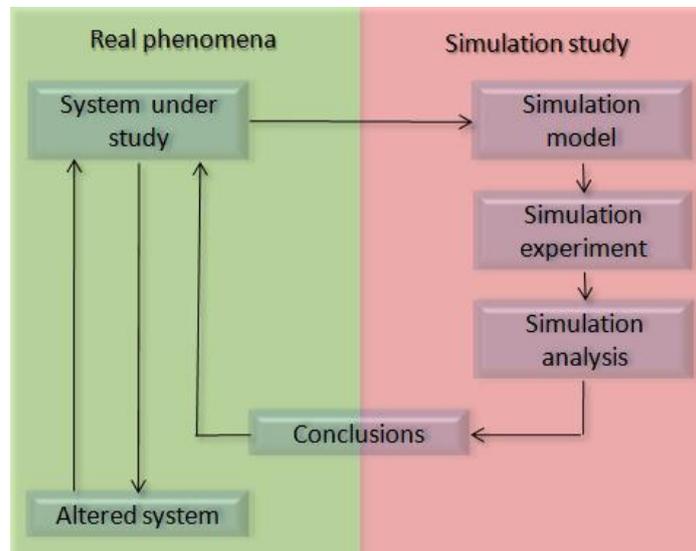
1. *From engineering point of view*: to understand the system in order to build it.
2. *From natural science viewpoint*: to understand more about nature.

Based on [9, 10, 13] simulation study is used, when:

- system or process is *impossible* or extremely *expensive* to observe in the real world;
- experimentation with a system is too *dangerous* or the system to be investigated doesn't exist yet;

<sup>2</sup> „Target“ (an adjective) – that is or may be a „goal“, desired goal.

- *time scale* of the dynamics of the system is too large and it takes millions of year to observe small changes in the system;
- some *variables* of the real system are inaccessible;
- *easy manipulation* with system parameters is necessitated;
- suppression of *disturbances* or *second-order effects* is needed.



**Figure 3:** Stages of simulation study

From the *Figure 3* it is evident, that before the simulation study will start, an identification and a formulation of a real problem is needed. Based on real system data, creation of a simulation model and modelling itself (i.e. time-evolution study of the system) are possible. Modelling also includes making of requirement model documentation. Simulation experiment begins from selection of an appropriate experimental design. The establishing of experimental conditions for run and the performing of simulation runs takes a place then. Simulation analysis is a final stage of simulation study. It is intended for evaluation and interpretation of simulation results. Conclusions, which are applied to system under study, come both from simulation study and real facts [9].

Recently, simulation studies based on mathematical models are carried out using different computer techniques. Then computer-implemented studies for exploring the properties of mathematical models are known as computer simulations [1]. *Humphreys* in his article “Numerical Experimentation” [14] claims that the computer simulation constitutes a new kind of scientific method, which is the connecting link between empirical experimentation and analytic theory. The reasons that lead to performance the simulation study are the same in a case of computer simulation. Computer simulation studies are often used when analytic solutions of formulated mathematical models are impossible or it is complicated to obtain them. According to *Hartmann* [1], computer simulation may also be helpful even if analytic solution for the target system is available. Visualizing the result of any kind of simulation on a computer screen is another advantage of it.

It is evident that implementation of simulation study on a target system has a number of benefits. But some dangers are also here. *Fritzson* in [10] features the following ones:

1. For user it is easy to forget or involuntary overpass limitations and conditions under which a simulation is valid. It leads to wrong conclusions from simulation study. In order to prevent it the comparison some results of simulation with known physical laws or experimental results from the real system are recommended.
2. Reaching the “Pygmalion effect”. In other words – to fall in love with model – forget that the model isn’t the real world but only represents the real system under certain conditions.
3. Forcing reality into the constraints of a model – the “Procrustes effect”.

### **1.4. Modelling and simulation in the textile industry**

From physical point of view a “textile” in general is an object, which can be described by the theories of classical physics and experimented with physical instruments. It is a physical three-dimensional body (extended in three-dimensions of space), which has a certain mass, location or position in space and is lasting for some period of time [15]. It is the subject of a study in an experiment and it is the object that could be referred to physical theories and laws. During last few years, the principles of modelling and simulation became to be popular in the textile industry also. For example, there is a tendency:

- to use image analysis for textile quality assessment;
- to carry out modelling and simulations of textile structures (to study various textile structures using computer simulation, to characterize the yarn unevenness by means of computer technologies);
- to aid the garment design with a computer;
- to study physical properties of textiles as a moisture and heat transfer using computational simulations. [16]

The development of textile's structure modelling and their physical properties simulation is linked to the advances in computer hardware and software on the one side, and necessity to solve more and more complicated phenomena associated either with production or application of textiles on the other side. It is impossible to do the complete summary of all computational methods, models and instruments used in textile engineering. Generally speaking, the design of textile structures and garments are often spoiled with the using of CAD system; the study of geometry properties of textile structures predominantly comprehends the image analysis instruments and methods for its evaluation; the study of physical properties of textiles tends to the solving of differential equations of motion and etc.

The subject of my interest is a fluid transport through the porous media, also through the nano-porous materials. The fluid flow through fibrous materials is a phenomenon that occurs in a range of technological processes and it is a subject of a wide interest in textile

industry for all the time. The textile industry encounters with this phenomenon during a lot of production and finishing processes. Examples range from dyeing processes, over filtration to high performance textiles with improved wearing comfort. Permeability is the physical parameter of primary interest during the comfort evaluation or final textile product testing. Invention of multilayer textile materials (for example, Gore-Tex fabrics in clothing) is based on an idea to combine various layers with different permeability to reach the maximal comfort with respect to the diffusion of water vapour outward and retention of external liquid droplets [17].

It was mentioned in [18], that a common requirement for understanding the transport properties of textiles is a detailed understanding regarding the transport of momentum through textile structures. This information is difficult to obtain experimentally and often the researches rely on “try and error” methods. During last couple of years, the study of fluid and heat transfer in porous structures was facilitated thanks to software Fluent. The software was developed by the company ANSYS, Inc. (USA). At present it is the most used commercial software based on a computation fluid dynamics (CFD) code that has been in use since 1983 and has been applied to a broad range of disciplines (e.g., aerospace, chemical, environmental, textile engineering, etc.). The solution of *Navier-Stokes equations* for fluid flow (Chapter 1.4.1), coupled with the energy and diffusion equations, is the principle of *Fluent* software. The Finite Element Method (FEM) is usually used for a solution of nonlinear partial differential equation as *Navier-Stokes equations* are. Fluent is also considered as a powerful approach to obtain insight into momentum transport within textiles. The few skilled works [18, 19], which have used the Fluent software for simulation of transport phenomena in textile structures, were founded.

By the way, traditional numerical simulations, represented by the *Navier-Stokes equations*, rely on the continuum approach [20]. But the approach would break down, when the length scale of the physical system decreases, concretely, when the Knudsen number became greater than about 0,2 (some authors as *Truesdell* and *Muncaster* [21] consider the value 1 as a threshold). *Knudsen number* ( $Kn$ ) is dimensionless parameter that determines the degree of appropriateness of the continuum model – the degree of rarefaction of gases encountered in a small flows through narrow channels and for an ideal gas it is:

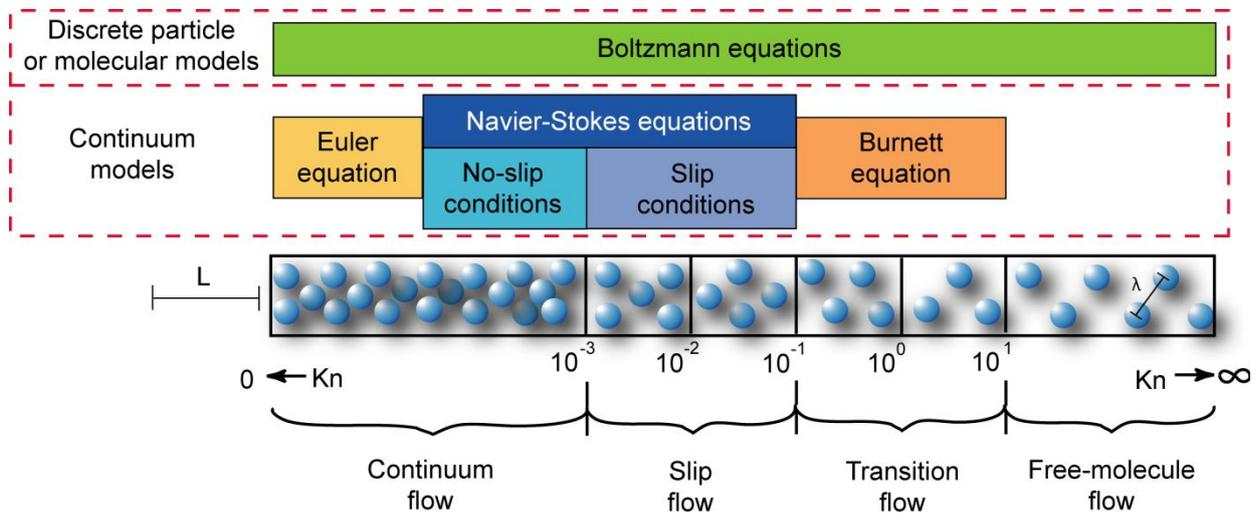
$$Kn = \frac{\lambda}{L} = \frac{k_B T}{\sqrt{2} \pi d_p^2 p L}, \quad (2)$$

where  $\lambda$  is a mean free path of molecules [ $m$ ];  $L$  is a length characterizing the geometry of flow, such as the diameter for a circular capillary, or the width of a pore, i.e. any microscopic dimension of interest [ $m$ ],  $k_B$  is a Boltzmann’s constant (approximately  $1.38 \times 10^{-23}$  [J/K]);  $T$  – temperature [ $K$ ];  $d$  – particle diameter [ $m$ ];  $p$  is a total pressure [ $Pa$ ].

From the Equation (2) it is evident: if the  $Kn$  is near or greater than one, the mean free path of a molecule is comparable to a length scale of the system or it is greater. The continuum assumption of fluid mechanics is no longer a good approximation. If we will consider the fluid flow through very small capillary pores, for  $Kn \geq 1$  intermolecular collisions are

become to be much less frequent than molecular interactions with solid boundaries. The intermolecular collisions can be ignored than. Flows under such conditions are termed *collisionless* or *free-molecular flow*. In this case discrete particle methods must be used instead of continuum approach.

As is shown in the *Figure 4*, only Boltzmann equation (Chapter 1.4.2), which is based on the discrete kinetic theory, is valid for the whole range of Knudsen number. As it was mentioned in [20], an alternative to continuum model is the molecular one, which recognizes the fluid as a swarm of discrete particles. Position, inertia and state of all individual particles are calculated here either deterministically or probabilistically at all times. During last few decades a large number of molecular models/methods, which consider individual particle dynamics based on a Boltzmann distribution at the temperature of interest, have emerged. Those methods are mesoscopic and include: molecular dynamic (MD), direct simulation Monte Carlo (DSMC), dissipative particle dynamics (DPD), smooth-particle hydrodynamics (SPH), Lattice gas cellular automata and Lattice Boltzmann model (LBM). Those methods are also used for the study of macroscopic hydrodynamics. They aren't based upon Navier-Stokes equations, but closely related to kinetic theory and Boltzmann equation. Those methods are mentioned in literature as promising candidates effectively connecting microscopic and macroscopic scales and enabling to study mesoscopic phenomena as a fluid transport in nanopores structures.



**Figure 4:** Different regimes of fluid flow and methods for their description depending on Knudsen number

During last few years, investigation of nanometric flow plays a crucial role in material science including textile engineering branch. Tendency to use lattice gas cellular automata for nanometric fluid flow modelling will be trashed out in chapters given below.

Next two chapters describe theoretical approaches, as the *Navier-Stokes equation* and the *Boltzmann equation*, useful for fluid flow modelling. Both methods characterize the same phenomenon but use the different principle for that. The *Navier-Stokes equation* presents macroscopic or continuum approach, where fluid flow is described by a finite number of

position dependent quantities such the mass density, the mean velocity, etc (see Chapter 1.4.1). In contrast to *Navier-Stokes equation* the *Boltzmann equation* uses microscopic approach. It characterizes the fluid flow using description of the dynamics of its individual particles (see Chapter 1.4.2).

### 1.4.1. Navier-Stokes equation

The *Navier-Stokes equation* is an equation describing the flow of incompressible Newtonian fluids<sup>3</sup>. The equation was derived by French engineer and physicist *Claude-Louis Navier* in 1827 and Irish mathematician and physicist *George Gabriel Stokes* in 1845 independently on each other. The detailed derivation of the Navier-Stokes equation is introduced for example in [22] and [23]. *Feynmann* in [24] describes in detail the essence of the equation.

According to *Feynmann* [24], to describe the motion of a fluid it is necessary to know the fluid properties at every point. At first we need to know vector and scalar fields of characteristics, which vary at every point of fluid and for any time. Those characteristics are density, pressure and velocity. *Feynmann* bases on the assumption:

- density and pressure determine the temperature at any point;
- density is a constant – fluid is essentially incompressible – it is expected, that variations of pressure are so small (or the velocities of flow are much less than the speed of sound wave in the fluid) that the changes in density produced thereby are negligible.

The interpretation of the essence of *Navier-Stokes equation* begins from an equation of state for the fluid which connects the pressure  $p$  to the fluid density  $\rho$  [24]:

$$\rho = const \tag{3}$$

If the fluid velocity is  $\mathbf{v}$ , then the mass which flows in a unit time across a unit area of surface is the component of  $\rho\mathbf{v}$  normal to the surface. Then the hydrodynamic equation of continuity is<sup>4</sup>:

$$\nabla \cdot (\rho\mathbf{v}) = -\frac{\partial\rho}{\partial t} \tag{4}$$

The Equation (4) expresses the conservation of mass for a fluid. According to the assumption ( $\rho = const$  - see Equation (3)) the equation of continuity becomes:

$$\nabla \cdot \mathbf{v} = 0 \tag{5}$$

---

<sup>3</sup> *Newtonian fluid* is a rheological model of a viscous substance, which is governed by Newton's law of viscosity. Rheological equation of Newtonian fluid is characterized by direct proportionality between strain rate and stress. The constant of proportionality here is known as viscosity.

<sup>4</sup> Symbol  $\nabla$  denotes the vector of differential operations  $\left(\mathbf{i}\frac{\partial}{\partial x}, \mathbf{j}\frac{\partial}{\partial y}, \mathbf{k}\frac{\partial}{\partial z}\right)$  containing unitary vectors  $\mathbf{i}$ ,  $\mathbf{j}$  and  $\mathbf{k}$  oriented along  $x$ ,  $y$  and  $z$  axes respectively.

From the Equation (5) it is evident, that the fluid velocity  $\mathbf{v}$  has zero divergence. Zero divergence means that the velocity doesn't change at a given point of the velocity vector field, it is a constant.

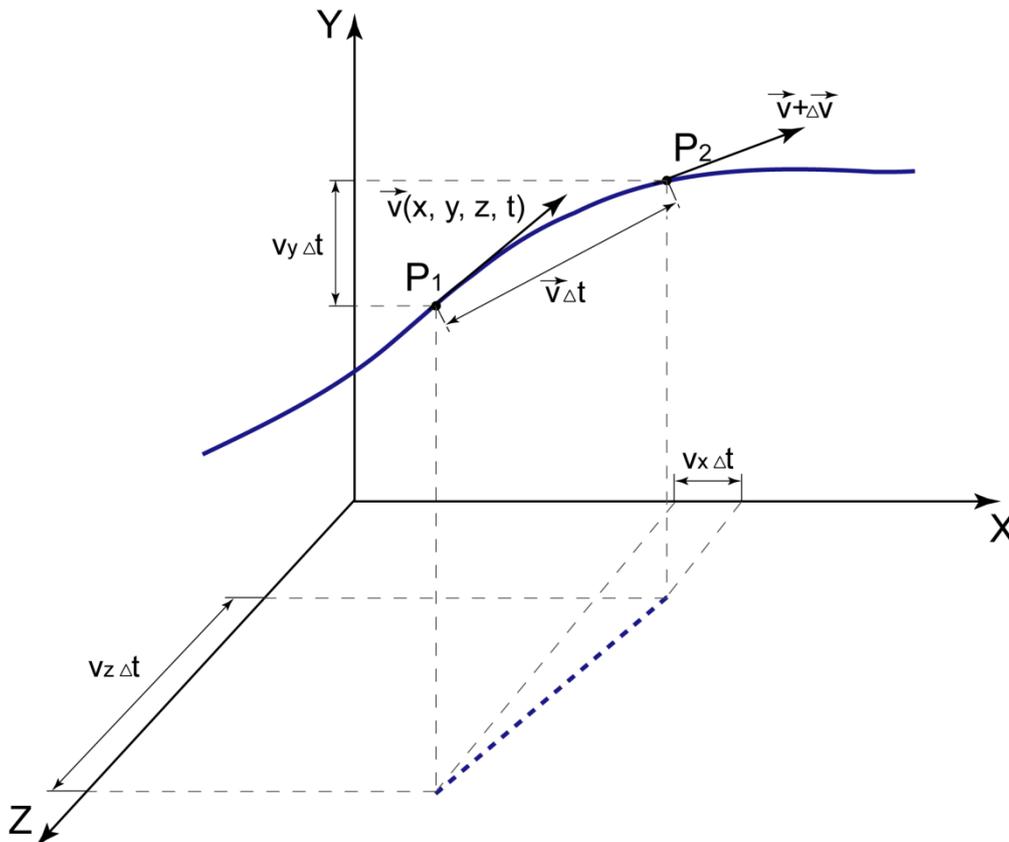
A second *Newton's law* tells how the velocity of the body changes because of the forces ( $F = ma$ ). Taking an element of unit volume and writing the force per unit volume as  $\mathbf{f}$ , we will get:

$$\mathbf{f} = \rho \mathbf{a} \tag{6}$$

The force density  $\mathbf{f}$  ( $f = \frac{F}{V}$ , where the  $V$  is volume) in an Equation (6) is the sum of three terms: pressure force per unit volume  $-\nabla p$  (consequence of the existence of pressure gradient); external forces like gravity etc. – when they are conservative force with a potential per unit mass  $\varphi$ , they give a force density  $\rho \nabla \varphi$ ; internal force per unit volume (consequence of the existence of shearing stress) – viscous force  $\mathbf{f}_{visc}$ . Then the equation of motion is:

$$\rho \mathbf{a} = -\nabla p + \rho \nabla \varphi + \mathbf{f}_{visc} \tag{7}$$

For the expression of acceleration *Feynmann* deals how fast the velocity changes for a particular pieces of fluid. If we will consider the movement of the drop of water in a small interval of time  $\Delta t$  from point  $P_1$  to  $P_2$  along some path, it will move by an amount  $\mathbf{v} \Delta t$  (see *Figure 5*).



**Figure 5:** The acceleration of fluid unit volume

If  $\mathbf{v}(x, y, z)$  is the velocity of the fluid unit volume at the time  $t$  at a position  $(x, y, z)$ , then the velocity of the same unit volume at the time  $t + \Delta t$  will be:

Time	Position of the fluid unit volume	Velocity
$t$	$P_1(x, y, z)$	$\mathbf{v}(x, y, z)$
$t + \Delta t$	$P_2(x + \Delta x, y + \Delta y, z + \Delta z)$	$\mathbf{v}(x + \Delta x, y + \Delta y, z + \Delta z, t + \Delta t)$ , where $\Delta x = v_x \Delta t; \Delta y = v_y \Delta t; \Delta z = v_z \Delta t$

From the definition of the partial derivatives (Taylor series):

$$\begin{aligned} \mathbf{v}(x + v_x \Delta t, y + v_y \Delta t, z + v_z \Delta t, t + \Delta t) &= \\ &= \mathbf{v}(x, y, z, t) + \frac{\partial \mathbf{v}}{\partial x} v_x \Delta t + \frac{\partial \mathbf{v}}{\partial y} v_y \Delta t + \frac{\partial \mathbf{v}}{\partial z} v_z \Delta t + \frac{\partial \mathbf{v}}{\partial t} \Delta t \end{aligned} \quad (8)$$

The acceleration  $\mathbf{a} = \frac{\Delta \mathbf{v}}{\Delta t}$  is:

$$\mathbf{a} = \left( v_x \frac{\partial}{\partial x} \right) \mathbf{v} + \left( v_y \frac{\partial}{\partial y} \right) \mathbf{v} + \left( v_z \frac{\partial}{\partial z} \right) \mathbf{v} + \frac{\partial \mathbf{v}}{\partial t} \quad (9)$$

Because  $\left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \equiv \nabla$  is a divergence, then:

$$\mathbf{a} = (\mathbf{v} \cdot \nabla) \mathbf{v} + \frac{\partial \mathbf{v}}{\partial t} \quad (10)$$

If the velocity at given point isn't changing ( $\frac{\partial \mathbf{v}}{\partial t} = 0$ ), then acceleration is zero. Putting the acceleration from Equation (10) into Equation (7) we will get:

$$\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v} = -\frac{\nabla p}{\rho} + \nabla \varphi + \frac{\mathbf{f}_{visc}}{\rho} \quad (11)$$

Equation (11) is a *general form of Navier-Stokes equation* for an incompressible fluid flow.

To find the solution of the *Navier-Stokes equation* of motion it is necessary to rearrange the Equation (11) by using the following identity from vector analysis:

$$\nabla(\mathbf{A} \cdot \mathbf{B}) = (\mathbf{A} \cdot \nabla) \mathbf{B} + (\mathbf{B} \cdot \nabla) \mathbf{A} + \mathbf{A} \times (\nabla \times \mathbf{B}) + \mathbf{B} \times (\nabla \times \mathbf{A})$$

As a special case, when  $\mathbf{A} = \mathbf{B}$ :

$$\nabla \mathbf{A}^2 = 2(\mathbf{A} \cdot \nabla) \mathbf{A} + 2(\mathbf{A} \times (\nabla \times \mathbf{A}))$$

$$\frac{1}{2} \nabla \mathbf{A}^2 = (\mathbf{A} \cdot \nabla) \mathbf{A} + \mathbf{A} \times (\nabla \times \mathbf{A})$$

So,  $(\mathbf{A} \cdot \nabla) \mathbf{A}$  corresponds to the  $(\mathbf{v} \cdot \nabla) \mathbf{v}$ , eventually:

$$(\mathbf{v} \cdot \nabla) \mathbf{v} = \mathbf{v} \times (\nabla \times \mathbf{v}) + \frac{1}{2} \nabla \mathbf{v}^2$$

Lets to define a new vector field  $\boldsymbol{\Omega}$ , as the curl of  $\boldsymbol{v}$ :  $\boldsymbol{\Omega} = \nabla \times \boldsymbol{v}$ . The equation of motion becomes:

$$\frac{\partial \boldsymbol{v}}{\partial t} + \boldsymbol{\Omega} \times \boldsymbol{v} + \frac{1}{2} \nabla v^2 = -\frac{\nabla p}{\rho} + \nabla \phi + \frac{\boldsymbol{f}_{visc}}{\rho} \quad (12)$$

The vector field  $\boldsymbol{\Omega}$  is called vorticity. If the vorticity is zero everywhere, the flow is irrotational.

If the fluid is “thin” (in the sense that the viscosity is unimportant) and an object of interest is the velocity field, than  $\boldsymbol{f}_{visc}$  and pressure can be eliminated from the Equation (12). Taking the curl of both sides of Equation (12) and taking into account that the curl of the gradient of scalar field is the zero vector ( $\nabla \times (\nabla \phi) = 0$ , where  $\phi$  is any scalar field) we will get:

$$\begin{aligned} \nabla \times \left( \frac{\partial \boldsymbol{v}}{\partial t} \right) + \nabla \times (\boldsymbol{\Omega} \times \boldsymbol{v}) + \frac{1}{2} \nabla \times (\nabla v^2) &= \nabla \times (\nabla \phi) \\ \nabla \times \left( \frac{\partial \boldsymbol{v}}{\partial t} \right) + \nabla \times (\boldsymbol{\Omega} \times \boldsymbol{v}) &= 0 \\ \frac{\partial (\nabla \times \boldsymbol{v})}{\partial t} + \nabla \times (\boldsymbol{\Omega} \times \boldsymbol{v}) &= 0 \\ \left( \frac{\partial \boldsymbol{\Omega}}{\partial t} \right) + \nabla \times (\boldsymbol{\Omega} \times \boldsymbol{v}) &= 0 \end{aligned} \quad (13)$$

Equation (13) obtained from *Navier-Stokes equation* together with the equations

$$\boldsymbol{\Omega} = \nabla \times \boldsymbol{v} \quad (14)$$

and

$$\nabla \cdot \boldsymbol{v} = 0 \quad (15)$$

describes completely the velocity field  $\boldsymbol{v}$  of the incompressible fluid. Equation (14) defines the vector field  $\boldsymbol{\Omega}$  and Equation (15) is a equation of continuity when the fluid density  $\rho$  is constant.

Is well known, the *Navier-Stokes equation* is analytically solvable only in a few cases of simple flows (as an example, stationary flows in simple channel – Poiseuille flow). In more complicated cases it is necessary to solve the equation numerically. The problem with a solution of the *Navier-Stokes equation* is caused by the  $(\boldsymbol{v} \cdot \nabla) \boldsymbol{v}$ , which is nonlinear and is quadratic in  $\boldsymbol{v}$ . Mathematicians have not yet proven that the solution always exists in three dimensions. The Clay Mathematics Institute has ranked the solution of the *Navier-Stokes equation* among seven major mathematical problems, so-called “Millennium problems” [25].

### 1.4.2. Boltzmann equation

Except *Navier-Stokes equation* there is another theoretical approach, which makes possible to describe the fluid flow phenomenon. It is the *Boltzmann equation*, also known as a *Boltzmann transport equation* or *Boltzmann kinetic equation*. It was devised by Austrian

physicist *Ludwig Eduard Boltzmann* in 1872. In contrast to the principle of *Navier-Stokes equation*, the *Boltzmann* one reflects the state of a fluid by means the state of many identical point particles confined to a spatial domain. The state of a fluid is described here at kinetic level using so called distribution function  $f$ .

According to *Kittel* [26] the *Boltzmann equation* is an equation for the time evolution of the distribution function  $f(\mathbf{r}, \mathbf{v})$  in a one-particle phase space<sup>5</sup>. Here  $\mathbf{r}$  and  $\mathbf{v}$  denote, respectively, the position and velocity vectors, they are elements of the phase space. In a general form the distribution function  $f(\mathbf{r}, \mathbf{v})$  is determined by the ratio:

$$f(\mathbf{r}, \mathbf{v}) d\mathbf{r} d\mathbf{v} = \text{number of particles in } d\mathbf{r} d\mathbf{v} \quad (16)$$

$f(\mathbf{r}, \mathbf{v}) d\mathbf{r} d\mathbf{v}$  is the average number of particles, which at time  $t$  have position  $(\mathbf{r}, \mathbf{v})$  lying within a volume element  $d\mathbf{r} d\mathbf{v}$ . Because particles move inside and outside of the volume element  $d\mathbf{r} d\mathbf{v}$  and collide with each other, the function will change over the time with a rate:

$$\frac{\partial f}{\partial t} = \left(\frac{\partial f}{\partial t}\right)_{\text{movem}} + \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \quad (17)$$

The Equation (17) is done according to assumption that the number of particles doesn't change. The effect of a time displacement  $dt$  on the distribution function is then:

$$f(t + dt, \mathbf{r} + d\mathbf{r}, \mathbf{v} + d\mathbf{v}) = f(t, \mathbf{r}, \mathbf{v}) \quad (18)$$

The Equation (18) is in accordance with *Liouville's theorem* of classical mechanics (i.e. if the volume element follows along the streams the distribution is conserved) in the absence of collisions. With collisions it is:

$$f(t + dt, \mathbf{r} + d\mathbf{r}, \mathbf{v} + d\mathbf{v}) - f(t, \mathbf{r}, \mathbf{v}) = dt \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \quad (19)$$

The total derivation of the function  $f(t, \mathbf{r}, \mathbf{v})$  over the time is:

$$df = \frac{df}{dt} dt = \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial \mathbf{r}} \frac{d\mathbf{r}}{dt} dt + \frac{\partial f}{\partial \mathbf{v}} \frac{d\mathbf{v}}{dt} dt = dt \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \quad (20)$$

Lets  $\mathbf{v}$  and  $\mathbf{a}$  denote, respectively, the velocity  $\frac{d\mathbf{r}}{dt}$  and the acceleration  $\frac{d\mathbf{v}}{dt}$ , then:

$$\frac{\partial f}{\partial t} + \mathbf{v} \frac{\partial f}{\partial \mathbf{r}} + \mathbf{a} \frac{\partial f}{\partial \mathbf{v}} = \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \quad (21)$$

or

$$\frac{\partial f}{\partial t} + \mathbf{v} \frac{\partial f}{\partial \mathbf{r}} + \frac{\mathbf{F}}{m} \frac{\partial f}{\partial \mathbf{v}} = \left(\frac{\partial f}{\partial t}\right)_{\text{coll}} \quad (22)$$

<sup>5</sup> Phase space is defined as a space, in which all possible states of a system are represented. One-particle phase space corresponds to the space of all possible states of the one particle.

The Equations (21) and (22) represent the *Boltzmann transport equation*. In abstract form the *Boltzmann equation* is often written as following:  $\frac{\partial f}{\partial t} = C[f]$ , where  $C[f]$  is a collision term, which is account as a result of particle interactions.

*Kittel* in [26] expresses the collision operator  $\left(\frac{\partial f}{\partial t}\right)_{coll}$  by the introduction of the relaxation time  $\tau(\mathbf{r}, \mathbf{v})$ :

$$\left(\frac{\partial f}{\partial t}\right)_{coll} = -\frac{f - f_0}{\tau} \quad (23)$$

Here  $f_0$  is the distribution function in thermal equilibrium state. After combination Equations (16), (21) and (23) the *Boltzmann transport equation* in the relaxation time approximation is:

$$\frac{\partial f}{\partial t} + \mathbf{v} \frac{\partial f}{\partial \mathbf{r}} + \mathbf{a} \frac{\partial f}{\partial \mathbf{v}} = -\frac{f - f_0}{\tau} \quad (24)$$

The following Chapter 2 describes Lattice Gas Cellular Automata whose nature reflects the Boltzmann transport equation.

## 2. MODELLING WITH CELLULAR AUTOMATA AND LATTICE GAS CELLULAR AUTOMATA

From computer science the study of certain phenomena suggests that there are computer systems that may be appropriate as models for microscopic physical phenomena. Cellular automata are now being used to model varied physical phenomena. *Fredkin* in his paper [27] wrote about cellular automata (CA) modelling:

*“The computer science approach to modelling physics with CA is qualitatively different from either theoretical or experimental physics, or from the kinds of abstract mathematical work that so often leads to progress in physics. The problem is that the study of cellular automata is both a theoretical and an experimental science. However, the experiments, which often produce results we did not anticipate, are not like physics experiments. They are the kind of experiments that never existed before the age of the computer.”*

*Richard Feynman’s* view of lattice-gases, as paraphrased by one of his co-workers, *Daniel Hillis* [28] was:

*“We have noticed in nature that behaviour of a fluid depends very little on the nature of the individual particles in that fluid. For example, the flow of sand is very similar to the flow of a pile of ball bearings. We have therefore taken advantage of this fact to invent a type of imaginary particle that is especially simple for us to simulate. This particle is a perfect ball bearing that can move at a single speed in one of six directions. The flow of these particles on a large enough scale is very similar to the flow of natural fluids.”*

It is necessary to describe the basic principles of Cellular Automata and Lattice Gas modelling for the purpose of this work. For that reason I will allow myself to present the description of the basic properties of Cellular Automata and Lattice Gas Cellular Automata in Chapters 2.1 – 2.4.

### 2.1. Historical overview: cellular automata and lattice gas automata

It seems currently to be quite impossible to survey the area of cellular automata in a whole range. Cellular automata have been invented independently for quite a number of times and as indicated in [29] for a wide variety of purpose and under different names: “tessellation automata”, “homogeneous structures”, “cellular structures”, “tessellation structures” and “iterative arrays”.

Cellular automata are coming from the time when a development of computer technique started. One admits commonly that cellular automata have been introduced by *John von Neumann*, the famous Hungarian mathematician, under the name “cellular space” in the end

of 1940's. Whereas other refer that cellular automata were introduced by *John von Neumann* and *Stanislaw Ulam* independently from *Konrad Zuse* [30, 31].

*Ulam* and *Neumann* were mathematicians working together on the U.S. Los Alamos project during the Second World War. They belonged to research team working on a development of modern computers. As it is mentioned in [32] *Ulam* liked to design pattern games for the Los Alamos huge computer. The first game was aimed at printing ever-changing patterns, which grew almost as if they would have been alive. The next game, developed by *Ulam*, constructed three-dimensional "recursively defined geometric objects". Each cell pattern from this kind of *Ulam's* games consisted of groups of cells creating different shapes in a space (square, triangular, hexagonal). These games were played on an infinite chessboard, i.e. on an infinite lattice. All changes of these cell patterns took part in discrete time steps. A fortune of particular cell state depended on states of its neighbouring cells. So *Ulam* constructed first cellular spatial games and he shared his skills in that with his co-worker *John von Neumann*.

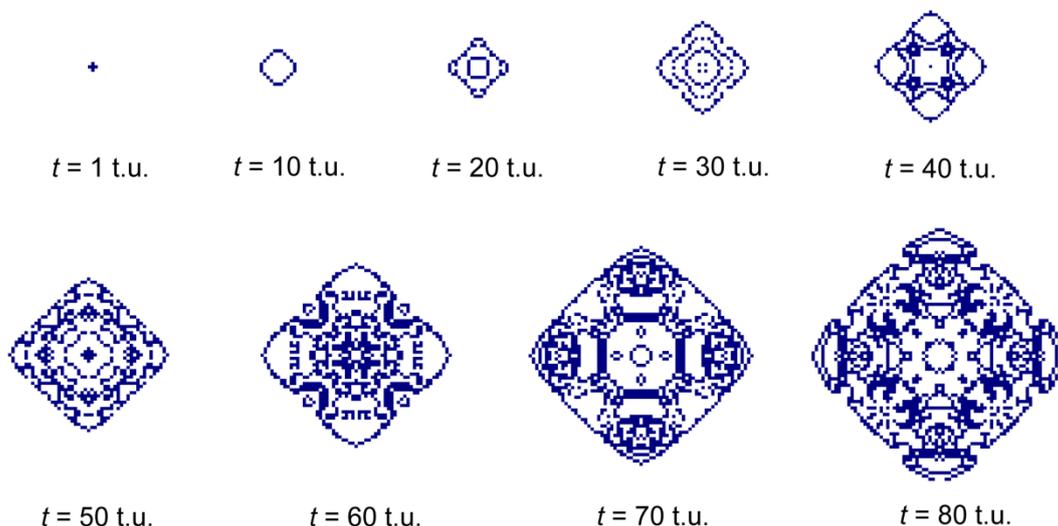
Thanks to *Goldstine* [32], who created a research team to work on problems in computers, communications, control, and time-series analysis in 1944, *Neumann* was introduced to electronic computing problems also. *Neumann* proceeded on design of Electric Discrete Variable Computer (EDVAC) in 1946. It was the first attempt to design physical automata ideas, first developed by *Post* and *Turing* at the end of 1930's. In that time *Neumann's* work included studies on the complexity that is required for a device or a system to be self-reproductive. *Neumann* was a pioneer in the study of a self-reproducing automaton based on a "system of non-linear partial differential equations, essentially of the diffusion type" and on algorithms of parallel computing [33, 34]. *Ulam's* ideas about an abstract space of cells, each of which is assign with a finite number of states, with local and uniform interactions among them found their usage in these *Neumann* studies. In the same time, independently on works of *Neumann*, *Zuse*, who was interested in numerical methods in mechanics, came with idea of parallel processing. But special historical circumstances forestalled the popularity of his work. His book named "Calculating space" was published only once in 1969 [35, 36]. Some of his formulations resemble the first and the most simple lattice gas models based on cellular automata method. The latter it has been proposed four years later by *Hardy*, *de Pazziz* and *Pomeau* and was well known recently as HPP model [31]. The most far-reaching vision of *Zuse* was that physical laws of the universe are discrete by nature, and that the entire universe is just the output of a deterministic computation of a giant cellular automaton.

It is mentioned in literature that two main pathways appeared for cellular automata development starting with *Neumann's* pioneering works. The first of them raised cellular automata, originally perceived merely as "toy" tools, for investigation and monitoring of serious biological systems. At least cellular automata penetrated into computer problems and dominated in this area for next few decades. A brief history of cellular automata in computer science and mathematics is presented in [37]. The path of cellular automata

development in the area of biology with connections to some physical problems will be traced in brief at the end of this subsection.

An excellent instance of cellular automata application in biology is the game called “Life” invented by *John Conway*. It was popularized among members of early computing community by *Martin Gardner* [38] in seventies. The game “Life” is a simple two-dimensional analogy of basic processes in living systems. It is based upon tracing temporal changes in a pattern, formed by sets of “living cells”. Each cell in a grid may be in either of two states: “alive” or “dead”. The state of each cell changes in time from one generation to the next one according to the update rule. This rule takes into account a state of a certain cell and states of its neighbours [32] in a similar way as it was indicated in former *Ulam’s* games. A system evolution over 80 time steps from an initial state is presented in the *Figure 6*. A “time step” (t.s.), also known as a “time unit” (t.u.), in contrast to the real time<sup>6</sup>, is a unit, needed for realization the one cycle of all operations in a simulation algorithm.

Many next researchers searched for cellular automata’s potential in modelling of biological systems [39, 40]. These works demonstrated that simple behaviour and functioning of live organisms can be modelled using cellular automata, where site values represent states of individual living cells or states of cell colonies. Short-range or contact interactions may lead to expression of “genetic characteristics” via the determination of cell colony patterns. It has been shown that simple update rules may lead to the formation of complex cellular patterns like in living cell colonies, plant and animal tissues.



**Figure 6:** Set of patterns obtained in game of “Life” for various time evolution steps  $t$  (courtesy of *Jakub Hrůza*) [17]

---

<sup>6</sup> Under the International System Measurement *second* is defined as a duration of 9 192 631 770 cycles of radiation corresponding to the transition between two electron spin energy levels of the ground state of the cesium (Cs) 133 atom. Time step could be equating to the time in itself, but it doesn’t have the same unit [86].

The set of theoretical studies and analysis of cellular automata's properties augured their occurrence in modelling of physical problems and especially in simulation of hydrodynamic phenomena. It has been already marked that in spite of simple update rules cellular automata can display complex behaviour, which is one of the most important conditions to use them as a simulation tool for the description of many-particle or collective physical phenomena. Partly discrete models, discrete with respect to time and space, were well known from biological applications of cellular automata since the end of sixties.

The first so-called classical Lattice Gas models appeared as theoretical ones, used for liquid-gas transition. They were structured nearly simultaneously in the late sixties and beginning of seventies [41]. A moment-conserving lattice gas model started to be an object of interest of hydrodynamics and statistical mechanics when *Kadanoff* and *Swift* proposed the first discrete-velocity model [42]. They created a version of *Ising* model in which positive spins acted as particles with momentum in one of four directions on a square lattice (see Chapter 2.3.2.1), while negative spins acted as holes. Particles were then allowed to collide with each other or to exchange their positions with holes if energy and momentum were exactly conserved [43]. The fully discrete model of hydrodynamics based on cellular automata conception, was firstly introduced by *Hardy*, *de Pazziz* and *Pomeau* [44]. This model nowadays is known as a HPP model. It led to a lot of interesting results, but due to using the square geometry of lattice it had limited applications because of its anisotropic behaviour. It was not refined until 1986, when *Frisch*, *Hasslacher* and *Pomeau* designed their own model, based on a triangular lattice. This model was called as the first FHP model. The detailed description of these models will be introduced in the Chapter 2.4.

During the last decades the development of the FHP model within the modelling of hydrodynamics led to the design of derivative models. In the next section examples of such few models and discussions of their usage for transport phenomena in porous materials will be given.

## **2.2. Specification of finite automata, cellular automata and lattice gas cellular automata**

The phrase "cellular automaton" usually indicates an infinite set of finite automata, which are interrelated in a specific manner. A lattice gas cellular automaton is a special case of cellular automaton. What do the terms finite automaton, cellular automaton, and lattice gas cellular automaton mean in general and in the realm of cellular automata? The definitions of the same are provided below.

### **2.2.1. Finite automata**

A "finite automaton" or "finite state automaton" (plural: automata) or "finite state machine" was firstly introduced and studied by *Cobham* in 1972 and has got the modern view in 1980's thanks to *Christol*, *Kamae*, *Mendes France* and *Rauzy*. In general, it is a class of

simplest mathematical model of processors, or special class of programming languages, that are characterized by having a finite number of states [45], which evolve in time and produce outputs according to rules depending on inputs [46].

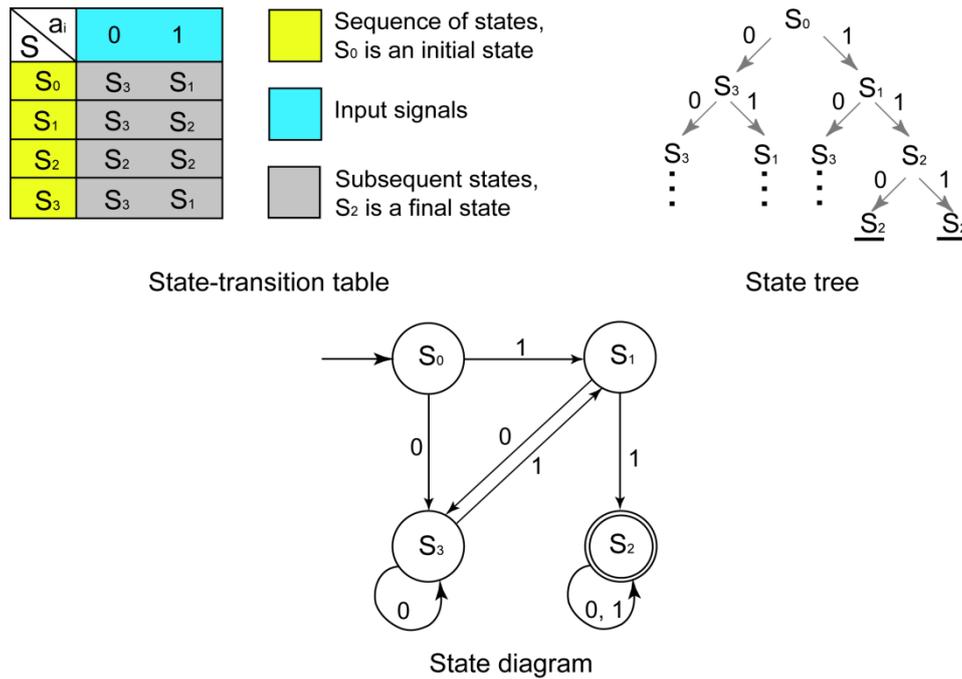
Similar definitions of finite automaton can be found in literature, which refers to principles of simulation, modelling and programming. Taking this view-point, a finite automaton (FA) is represented formally by the five-tuple  $FA = (S, S_0, \Sigma, \delta, F)$ , where:

- $S$  – is a finite, non-empty set of states (also known as a *state space*);
- $S_0$  – is an *initial state*, an element of  $S$ ;
- $\Sigma$  – is a finite, non-empty set of possible *input signals* (the set of input symbols or input alphabet);
- $\delta$  – is a *state-transition function*;
- $F$  – is a set of final or accepting states of  $FA$ , also is a subset (possibly empty) of  $S$  [47, 48].

The state-transition function drives the work of finite automaton and specifies, for each state and input alphabet, the next state the automaton will enter. For a given current state and a given input signal, if an automaton only jumps to one and only one state, then it is a *deterministic automaton*. Another type of automaton is a *non-deterministic finite automaton*. Here, after reading an input signal, automaton may jump into any of number of possible states driven by its transition relation. The most standard variant describes bellow is the deterministic finite automaton.

Three possible methods of finite automata representation (here it is the deterministic finite automaton) are shown in *Figure 7*:

- the *state-transition table* determines an initial state  $S_0$ , subsequent states  $\{S_1, S_2, S_3\}$ , final state  $S_3$  and state-transition function  $\delta$ ;
- the *state tree* is presented using original roots, which arise from the initial state  $S_0$ . The number of links that come out from each cusp of the tree is equal to the total number of input/output signals. Successors of each state are created according to the input signals, using the state-transition function  $\delta$ .
- the *state diagram* is consists of vertices, which agree with the state of automaton. Links indicate the possible transitions between all possible states. Here the arrow before the cusp with the initial state  $S_0$  denotes the start of the calculation of the finite automaton. Two circles that surround the state  $S_2$  mean that the state of the automaton is the final (accepting) one.



**Figure 7:** Finite automaton represented using classical methods: state-transition table, state tree and state diagram [17]

The computation of finite automaton on inputs  $a = a_1, a_2, \dots, a_n$  (where  $a_i \in \Sigma$  for all  $1 \leq i \leq n$ ) is the sequence of states  $S = S_0, S_1, \dots, S_n$  (where  $S_i \in S$  and  $S_0$  is the initial state):

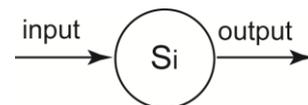
$$S_i = \delta(S_{i-1}, a_i) \text{ for all } 1 \leq i \leq n \tag{25}$$

For instance, for the input signal  $\{0,1,0\}$  the automaton presented in *Figure 7* begins in  $S_0$ , reads the input signal  $a_1 = 0$  and goes to the state  $S_1 = \delta(S_0, 0)$ . Then reads input signal  $a_2 = 1$  and goes to the state  $S_2 = \delta(S_1, 1)$ , finally it ends up in  $S_2 = \delta(S_2, 0)$  after reading the last input signal  $a_3 = 0$ . Formally this computation can be written as:

$$S_2 = \delta\left(\left(\delta\left(\left(\delta(S_0, 0), 1\right), 0\right), 0\right), 0\right) \tag{26}$$

So, the operation of the finite automaton can be easily displayed as is shown in *Figure 8*.

An output signal of a finite automaton can be used as an input signal for another finite automaton – it is the basic principle of cellular automata (see Chapter 2.2.2). The term “individual automaton” is used instead of “finite automaton” in the realm of Cellular Automata and Lattice Gas Cellular Automata models [46]. This notation will be followed hereafter.



**Figure 8:** Basic principle of a finite automaton operation

### 2.2.2. Cellular automata

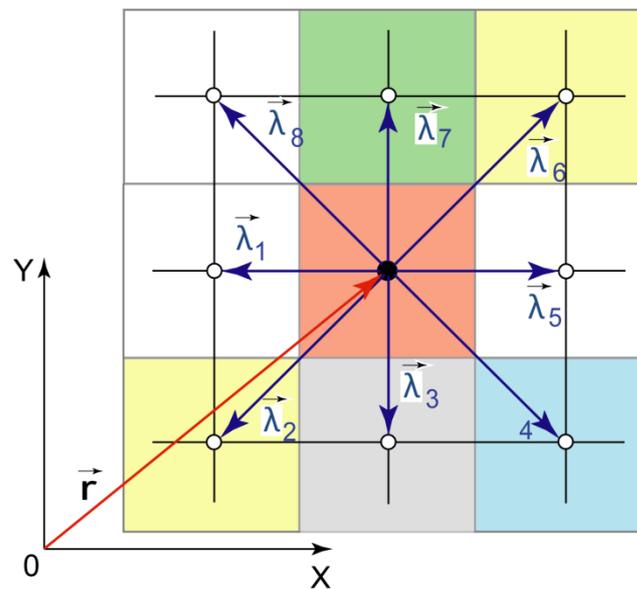
According to *Wolfram* [29], “cellular automaton” (plural: cellular automata – CA) is defined as a “...mathematical idealization of physical system in which space and time are discrete and physical quantities take on a finite set of discrete values”. Cellular automaton consists of:

- a set of identical sites (“cells”) located in a regular and uniform lattice (or “array”), usually infinite;
- each cell holds a finite number of discrete states. A set  $\mathcal{S}(\mathbf{r}, t) = \{S_0(\mathbf{r}, t), S_1(\mathbf{r}, t), \dots, S_n(\mathbf{r}, t)\}$  of Boolean variables (where  $S_i(\mathbf{r}, t)$  is a single bit of information) is attached to each site of a lattice by position vector  $\mathbf{r}$  and creates the local state of each cell at the time steps  $t = 0, 1, 2, \dots, n$ .
- states of all cells of CA are updated simultaneously at discrete time steps according to principles used in finite automaton (see Chapter 2.2.1);
- changes of states are governed by update rules (in finite automata it is also known as a state-transition function), which can be deterministic or non-deterministic, but always uniform in space and time;
- rules for evolution of a cell depend generally on a local neighbourhood of cells around it. The update rule  $\delta = \{\delta_1, \delta_2, \dots, \delta_n\}$ , which specifies the time evolution of the states  $\mathcal{S}(\mathbf{r}, t)$ , in general can be defined in following way:

$$S(\mathbf{r}, t + 1) = \delta(S(\mathbf{r}, t), S(\mathbf{r} + \boldsymbol{\lambda}_1, t), S(\mathbf{r} + \boldsymbol{\lambda}_2, t), \dots, S(\mathbf{r} + \boldsymbol{\lambda}_k, t)),$$

where  $\mathbf{r} + \boldsymbol{\lambda}_i$  ( $i \in \{1, 2 \dots k\}$ ) designate the cells belonging to a given neighbourhood of cell  $\mathbf{r}$ . If  $\mathbf{r}$  is a location of the certain cell, then  $\mathbf{r} + \boldsymbol{\lambda}_1, \mathbf{r} + \boldsymbol{\lambda}_2, \dots, \mathbf{r} + \boldsymbol{\lambda}_k$  are locations of its neighbours (see *Figure 9*). So, the new state of a cell having the location  $\mathbf{r}$  at time  $t + 1$  is only a function of its previous state in  $\mathbf{r}$  and states in neighbour locations  $\mathbf{r} + \boldsymbol{\lambda}_i$  at time  $t$  [49].

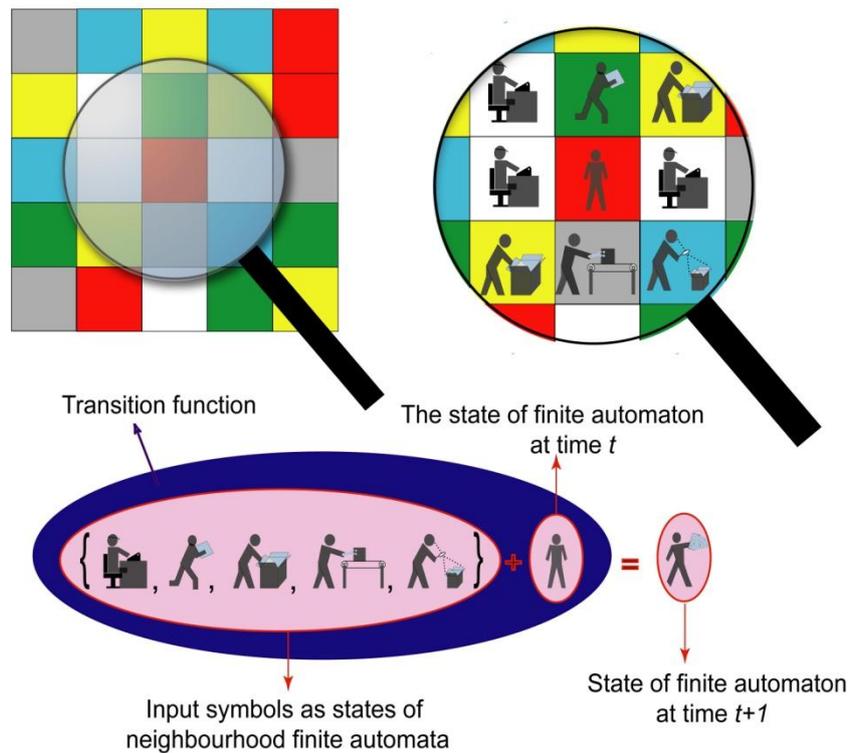
In the context of cellular automata, the term “neighbourhood” was first used. In cellular automata neighbourhood is usually created by cells surrounding a central cell with the position  $\mathbf{r}$ . Neighbourhood is done by the lattice geometry. More about neighbourhood types in accordance to lattice geometry is presented in Chapter 2.3.2.2.



**Figure 9:** Graphical explanation of the cell having the position  $\mathbf{r}$  and its neighbour cells located in a regular square lattice

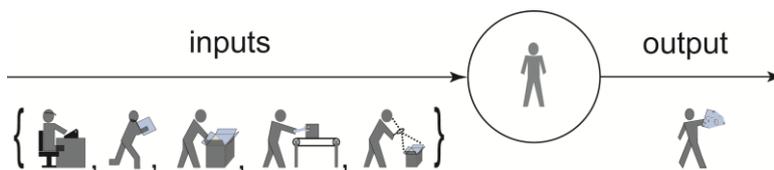
Therefore, a cellular automaton can be represented as a set of synchronized identical individual automata, which exchange their states with predefined neighbourhoods in accordance to an update rule, which is the same for all cells (i.e. finite automata, which comprise a cellular automaton) in a particular model [46]. Purposely, this definition does not contain any reference to the geometrical structure of a lattice, as it is not important to know the distances or angles between neighbours. However, it may be noted, that all individual automata in a cellular automaton are identical and create a homogeneous structure having uniform internal structure and obeying the same evolution and connection rules, except those, which are on boundaries. Such a cellular automaton can be presented as is shown in *Figure 10*.

The one evolution time step of the one cell for two-dimensional cellular automaton is illustrated in *Figure 10*. Different colours of cells in *Figure 10 (a, b)* represent various states of those cells at time  $t$ . Let us suppose, cells of the cellular automaton has seven possible states, all of them a evident from the *Figure 10 (c)*. If the central red cell will designated as a team-manager, and men in neighbour cells will perform during one time step the certain five activities, than in a next step at time  $t + 1$  team-manager will get the definite honorarium for his team. From cellular automata point of view, a person in the central red cell is an individual automaton (see *Figure 11*), collected information about activities of his employees and designated their financial state depending on their diligence. If the newly acquired state of the central red cell will marked with the purple colour, the cell will change its colour from red to purple in the next step at time  $t + 1$ . The alteration of cell states takes place synchronously for all cells in the lattice. Because CA must be also state-homogeneous, the state “manager” may appear in any cell of the lattice in a same way as any other activity or a state.



**Figure 10:** Graphical interpretation of two-dimensional cellular automaton: **a** – general appearance of a regular lattice, **b** – detailed configuration of neighbourhood cells of reference cell, **c** – application of a transition function and updating the state of the cell at time  $t+1$  [17]

In this session the cellular automaton based on a regular lattice was described. But there are also cellular automata, where cells are positioned randomly. Random connection of cells was proposed by *Richard Feynman* [28].



**Figure 11:** Cell of the cellular automaton as a individual automaton: states of the neighbour cells are inputs, the new state as an output of the automaton [17]

### 2.2.3. Lattice gas cellular automata as a special case of cellular automata

*Wolf-Gladrow* mentioned in [31]: “despite of their simple update rules cellular automata can display complex behaviour which is a prerequisite to use them as a simulation tool for physical (biological, chemical,...) phenomena like, for example, fluid flow”. The cellular

automaton that was able to simulate the fluid flow phenomena got the name “lattice gas cellular automata” (LGCA).

Lattice gas cellular automaton accounts itself as relatively new and promising method for the study of dynamic phenomena, which are often nonlinear and usually described by partial differential equations. Lattice gas cellular automata provide the numerical solution of those equations or enable the qualitative analyse of complicated physical tasks at that area at least. The field of LGCA started in 1973. In papers published in 1973 and 1976 *Hardy, de Pazzis* and *Pomeau* introduced the first lattice gas cellular automata named after their initials HPP model. Due to inappropriate lattice geometry, the HPP model proved to be highly anisotropic. More about that will be presented in the Chapter 2.3.2. The paper of *Frisch, Hasslacher* and *Pomeau* [50] in 1986 showed that phenomena based on a principle of billiard game with collisions that conserve mass and momentum, in the macroscopic limit leads to Navier-Stokes equation when the underlying lattice owns sufficient symmetry in a two-dimensional case. It was found that hexagonal lattice meets the condition of symmetry. So, LGCA became to be used for the simulation of fluid dynamics. The principles of molecular dynamics reflected in the Boltzmann equation (see Chapter 1.4.2), properties and abilities of cellular automata (see Chapter 2.2.2) have been joined together into the LGCA model.

Detailed description of the LGCA, definition, basic properties and types of the LGCA are presented in the Chapter 2.3.

### **2.3. Principles of lattice gas cellular automata**

As it was mentioned earlier in [50], the points of view from which a fluid can be described are, molecular, kinetic, and macroscopic. As it was presented in the Chapter 1.4.1 the detailed behaviour of a fluid at continuum macroscopic level is provided by partial differential equations, e.g., *Navier-Stokes equations*. Some other numerical techniques, such as, finite-difference and finite-element methods, are used for transforming a continuum system into a discrete one [51].

The Lattice Gas models based on Cellular Automata are newer compared to numerical methods mentioned above. These models make possible to describe the behaviour of fluid systems at a molecular level under various microscopic conditions. These models are based on detailed information about individual particles, such as their positions, masses, and velocities and they provide outputs in terms of molecular dynamics. Thus, lattice gas models entered into the history as an alternative for modelling fluid systems.

From the molecular theory developed in the last century it is known that individual molecules in crystals fluctuate around their locations in equilibrium state. Only occasionally they do jump out of their locations, such events are considered as fluctuations. These jumps occur due to their collisions with other molecules, when the system is shifted from its equilibrium state by some agent. A remarkable idea was to consider that a fluid has a

structure similar to a crystal and that every liquid molecule sits at some fixed point, having the same number of neighbouring sites at a definite distance. These sites are either empty or occupied by a molecule [52]. These spatially organized patterns of molecules are in accordance with a term ‘lattice gas model’. Different types of lattice gas models were proposed for description of a simple liquid<sup>7</sup> behaviour. There are two distinct basic lattice gas models mentioned in literature: non-interacting and interacting.

The non-interacting lattice gas is mentioned in *Kittel’s* book [26]. This model is represented by a set of  $N$  non-interacting atoms distributed over  $N_0$  lattice cells. Each cell is either occupied by one particle or empty. This system does not have any kinetic energy or any energy due to particle interactions. In spite of that, it found its application in statistical physics because non-interacting lattice gas model provides a correct shape of the ideal gas state equation where the pressure is obtained as a partial volume derivative of the system entropy.

The non-interacting lattice gas models together with cellular automata possibly helped to create the interacting lattice gas models – Lattice Gas Cellular Automata model.

According to *Rivet* and *Boon* [46] Lattice Gas Cellular Automata belong to the general class of cellular automata, thus sharing features characteristic to that class:

(i) Being one of the cellular automata, lattice gas cellular automata consist of identical individual automata which are tied geometrically to the nodes of a Bravais lattice, situated in an Euclidean space of dimension  $D$ . Individual automata are also called “nodes” in the purview of lattice gas cellular automata.

(ii) Instantaneous state of lattice gas cellular automata depends on the states of all individual automata. Each its individual automaton can inherit any one of the  $2^b$  states, where the quantity  $b$  represents the number of channels. Channels are links between neighbouring lattice nodes, i.e. neighbouring individual automata and they exactly copy the geometry of the lattice. In LGCA models each channel may either be occupied by a fictitious particle or remain empty and so, it has two possible states of existence. Thus, the state of an individual automaton can be interpreted as a set of states of channels, which connect the individual automaton with neighbouring ones. Consequently, information about the channel’s occupation corresponds to signals fed to individual automata.

(iii) The elementary evolution process of lattice gas cellular automata takes place in regular discrete time steps and consists of two distinct phases of evolution:

- *collision phase* – it is the first evolution step. During this phase, each individual automaton takes the new post-collision state depending on input signals and collision

---

<sup>7</sup>From fluid dynamics, *simple liquid* (fluid), is also known as a *Newtonian liquid* (fluid) – is a liquid in which the state of stress  $\tau$  [Pa] at any point is proportional to the time rate of strain at that point and the proportionality factor  $\mu$  [Pa\*s] is the viscosity coefficient:  $\tau = \mu \frac{dv}{dy}$ , where  $\frac{dv}{dy}$  is the velocity gradient perpendicular to the direction of shear or equivalently the strain rate [ $s^{-1}$ ].

rules. Input signals are obtained from neighbour individual automata and they contain information about the states of neighbour individual automata. The collision rules are the same for all individual automata and do not depend on their position. New states of individual automata generate output signals for the next evolution step;

- *propagation phase* – is the second evolution step. During this phase output signals of every individual automaton are conveyed to its neighbouring ones, i.e. neighbouring nodes, along the channels. Thus, these signals become a part of the input signals for its neighbours at the next time step. It is necessary to emphasise, that all changes in each individual automaton of the lattice gas cellular automata transmit output signals simultaneously.

The detailed description of the following properties and principles of LGCA, as a discretization of space and time, evolution rules are presented in the Chapters 2.3.1 – 2.4.2.

### 2.3.1. Discretization of space – basic methods. Grid generation

Over the years, many discretization algorithms or methods have been proposed. They have been developed due to various needs. In mathematics, discretization concerns the process of transferring continuous models and equations into discrete counterparts [53]. In physics, the discretization is defined as a substitution of a continuous media (continuum) by a *system of discrete points*, where different parameters of a related domain of the continuum are settled [54].

Discretization of space is an essential step to simplify continuous problems. As a result, the necessity to solve the partial differential equations transforms then into the solution of differential or algebraic equations only [54]. For example, *Navier-Stokes equation*, introduced in Chapter 1.4.1, is very difficult to solve using pen and paper, or analytically. For continuous problems and such types of analytical description, in the last 40 years, a brand-new computational approach was developed. Here, the complicated domain or a space is broken down into small pieces, each more simple to analyze. Hereby, the values at every of the infinite number of points of interest are reduced to the discrete set of values, which are finite.

So, in connection with physical definition of discretization, we are interested here in discretization methods, which direct to the formation of a set of discrete points, called nodes, and usually used in solution of physical continuous problems. The common discretization methods are:

- *Finite differences method (FDM)*, which is used to obtain an approximate solution of partial differential equation governing the behaviour of physical system by using

neighbouring points. The regular grid<sup>8</sup> is imposed on the physical domain. The approximation of derivative of an unknown quantity at a grid point takes place then. This approximation is given by the ratio of the difference of the unknown quantity at two neighbour points and the distance between grid points.

- *Finite volume method (FVM)* includes the splitting of a physical space into small volumes and subsequently the integration of the partial differential equations over each of volumes. Then the changes through the surface of each volume (fluxes at the surface of each finite volume) are approximated as a function of the variables in neighbouring volumes.
- *Finite elements method (FEM)* also splits up the space into small pieces. Each of the pieces is called an element. Compare with FDM, the FEM is an approximation of the solution of differential equation. Unlike the previous method, a grid point<sup>9</sup> exchanges the information with all the grid points which it shares an element [55].

So, the principle of all the above-mentioned methods is a splitting of the continual space onto a grid and thus obtaining the finite number of points in space and in time subsequently, at which variables are calculated. Adjacent points then are used to calculate derivatives. The discretization of a geometrical domain into small simple shapes (points, volumes, elements) is mentioned in literature as a “grid” or “mesh generation” [56]. During the grid generation the next criteria influence the grid geometry:

- *The local density of points* – the higher density is elected, the more accurate the solution is, but the computation takes more time.
- *The smoothness of the point distribution* – large variations in grid density or shape can cause numerical diffusion and as a result lead to inaccurate results or instability. The elements of the grid should not be overlapped ( $\sum S_{element} = S_{area}$ ) [57].
- *The shape of created grid elements* – elements of the grid should avoid both very sharp and flat angles; shapes of grid elements may cause serious numerical problems, etc. [58].

Three types of grids are distinguished in literature: structured, unstructured and hybrid. Their characteristics are typified in the *Table 1*.

The next part of the chapter is concerned with a study of lattices based on the structured grids – their types, characteristics, advantages and disadvantages in connection with their usage in lattice gas cellular automata modelling.

---

<sup>8</sup>*Grid* (also called *mesh*) is defined as a complex of elements discretizing the simulation domain with the aim of construction a discrete version of the original partial differential equations. In two-dimensional domain it is triangular or quadrilateral grid, in three-dimension it is tetrahedral or hexahedral [58].

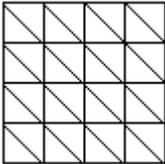
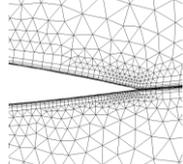
<sup>9</sup>Grid point (also called node) is a place, where elements are connecting.

### 2.3.2. Discretization of space in LGCA model

In order to describe natural systems accurately on an ordinary scale, models based on cellular automata require an approximation of the Euclidean geometry as closely as possible. For that reason different types of lattices (see Chapter 2.3.2.1) are used for a space discretization in LGCA models.

Each physical model (LGCA model in our case) that is defined on a lattice is opposed to the continuum space model and is known as the lattice model. Within the context of cellular automata or lattice gas cellular automata the term “lattice” is used rather than “grid” or “mesh”. Though in Czech language the meaning of all these terms is the same, in English it is referred to different definitions.

*Table 1: Characterization of different types of grids [57, 58]:*

<i>Characteristics</i>	<i>Structured (regular) grid<sup>10</sup></i>	<i>Unstructured grid<sup>11</sup></i>	<i>Hybrid grid</i>
The appearance			
Splitting the domain	Into regular grid elements	Is based on a density function that is defined by the input geometry or the numerical requirement	As a first step – splitting the domain into non-regular domain and then decomposing each such domain by regular grid.
Coordination number $b$ <sup>12</sup>	$b$ is constant, connectivity <sup>13</sup> can be calculated	Arbitrary	Arbitrary
Geometric flexibility	Lack	Greater	The highest
Generation intensity	Simple and fast	Harder and slower	The most hard
The “costs” associated with	Less computer memory is needed	Expensive in time, highest memory	The same as unstructured grid has

<sup>10</sup> A *regular grid* is a tessellation of the Euclidean plane by congruent rectangles or a space-filling tessellation of rectilinear parallelepipeds [87].

<sup>11</sup> An *unstructured grid* is a tessellation of a part of the Euclidean plane or space by simple shapes, such as triangles or tetrahedral, in an irregular pattern [88].

<sup>12</sup> *Coordination number  $b$*  is one of the important characteristics of lattices. According to [89] coordination number is the number of direction vectors; in [90] it is defined as the total number of neighbours of a given lattice node.

<sup>13</sup> Usually, *connectivity* of a grid or lattice characterizes the connection of its vertices and is defined as a total number of links that meet in a node [46]. Connectivity is a property of the lattice which is described by the value of coordination number.

usage		requirements are needed, the resulting linear system is hard to solve	
Application field	Problems with simple domain and smooth changes in solution; simulation based on cellular automata model (including LGCA model).	Finite element method and three-dimensional problems	Hydrodynamic studies especially unsteady flow involving multiple objects moving toward or away from each other – usually are solving by mathematical discretization methods [59]

In mathematics, a lattice in a  $n$ -dimensional Euclidean space is defined as a *discrete subgroup* of  $\mathbf{R}^n$  which spans the real vector space  $\mathbf{R}^n$ . Every lattice node in  $\mathbf{R}^n$  can be generated here from a *basis*<sup>14</sup> of the *unit vectors*  $\mathbf{e}_i$  by forming all linear combinations with *integer coefficients*  $a_i$  [60]:

$$\mathbf{r} = \sum_{i=1}^n a_i \mathbf{e}_i \quad (27)$$

In physics, lattice is a regular, periodic configuration of points, particles, or objects throughout an area or a space [61]. In materials science and solid-state physics, lattice is engaged as synonym for a crystalline structure and presents the arrangement of atoms or molecules in a crystalline solid. Contrary of the grid, lattice is usually viewed as a regular tiling of a space by primitive cell. Because the primitive cell is a minimum cell corresponding to a single lattice point of a structure with translational symmetry, lattice can be characterized by the geometry of its primitive cell. One of the characteristic properties of the cell geometry is the number of lattice nodes (sites) directly connected to a single lattice point. This number is known then as a *coordination number*  $b$  of the lattice.

The overview of primitive cell's geometry and appropriate lattices are presented in the part "Bravais lattices". More attention is given to Bravais lattices, which are being used in LGCA modelling (see Chapter 2.3.2.1).

### ***Bravais lattices***

Lattices and their symmetries were studied for the first time by *M.L. Frankenheim* in 1840's. He has found fifteen types of lattices. A few years later, *Auguste Bravais*, the French physicist, who is well known thanks to his work in crystallography, pointed out that two of

---

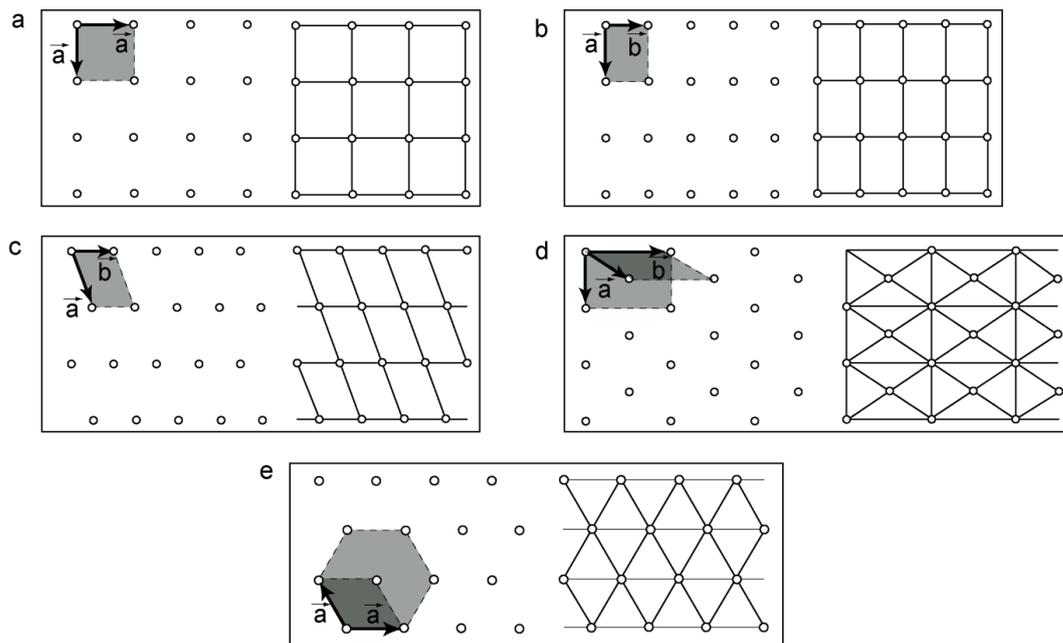
<sup>14</sup> *Basis* is a set of vectors that, in a linear combination, can represent every vector in a given vector space, and such no element of the set can be represented as a linear combination of the other. So, basis is a linearly independent spanning set [91].

the *Frankenheim* classes contained identical lattice, and that there are five two-dimensional and only fourteen three-dimensional lattices in crystalline system, which distinct from each other by the geometry of primitive cells. [62, 63] Now it is possible to see that all *Bravais lattices* fall within the set of structured grids.

Two-dimensional *Bravais lattices* are (see *Figure 12*):

- square;
- rectangular;
- oblique;
- centered rectangular (rhombic);
- hexagonal.

Today the definition of *Bravais lattice* is following: it is an infinite set of points generated by a set of discrete translation operations. As it was mentioned by *Rivet* in [46] due to the finite capacity of our computers the lattice in LGCA is only a subset (the finite number of lattice nodes) of the relevant *Bravais lattice*.



**Figure 12:** Two-dimensional *Bravais lattices*: **a** - square, **b** - rectangular, **c** - oblique, **d** - centered rectangular, **e** – hexagonal [17]

Lattice Gas Cellular Automata use two types of *Bravais lattices* for the space discretization:

- *square Bravais lattice* – was used in a first simple LGCA model, known as HPP<sup>15</sup> model;

---

<sup>15</sup> HPP model was the first lattice gas cellular automata. It is called after its autors: Hardy, de Pazzis and Pomeau

- from 1986 the *hexagonal Bravais lattice* was applied in a basic FHP model. Nowadays, the hexagonal lattice is the main one, which is being used in two-dimensional LGCA models (in a group of FHP models) and a *face centered hypercube lattice* in three-dimensional LGCA models.

Definitions and properties of square and hexagonal lattices and also their usability for a space discretization in LGCA modelling are described below.

### 2.3.2.1. Geometry of square and hexagonal lattices

As was mentioned before, the square and hexagonal lattice (equivalent is triangular lattice) are two of the five two-dimensional *Bravais lattices*, which are being used for a space discretization in LGCA models because of their high symmetry. The most common types of square lattice regarding to its orientation are: *upright square lattice* and *diagonal square lattice*, which differ by an angle of  $45^\circ$  (see *Figure 13*).

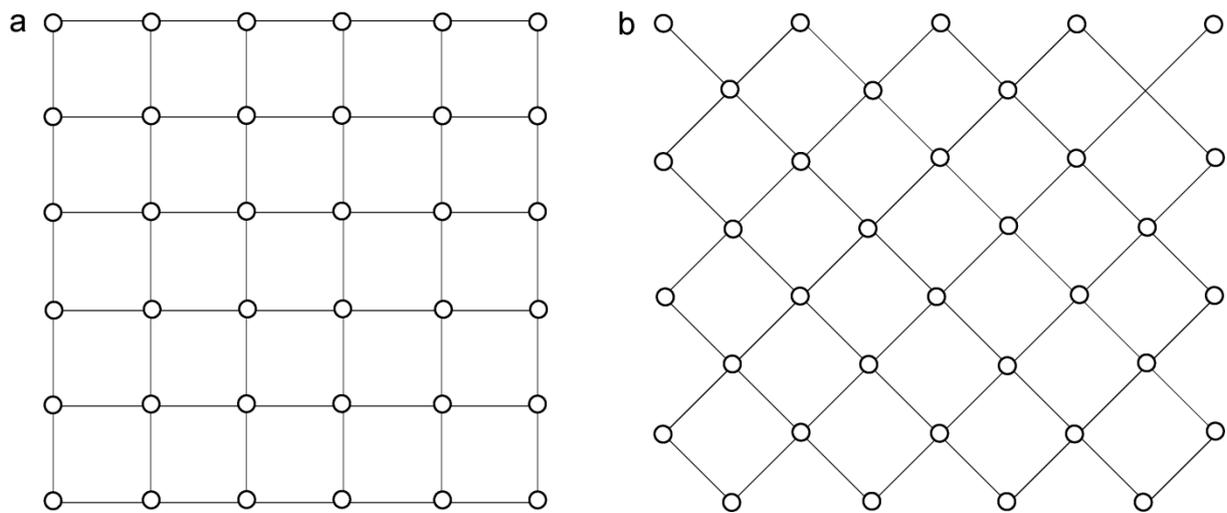
In a first LGCA model, in the HPP model, the upright square lattice was used. The set of direction vectors here was the following one:

$$\{(1,0), (-1,0), (0,1), (0,-1)\}$$

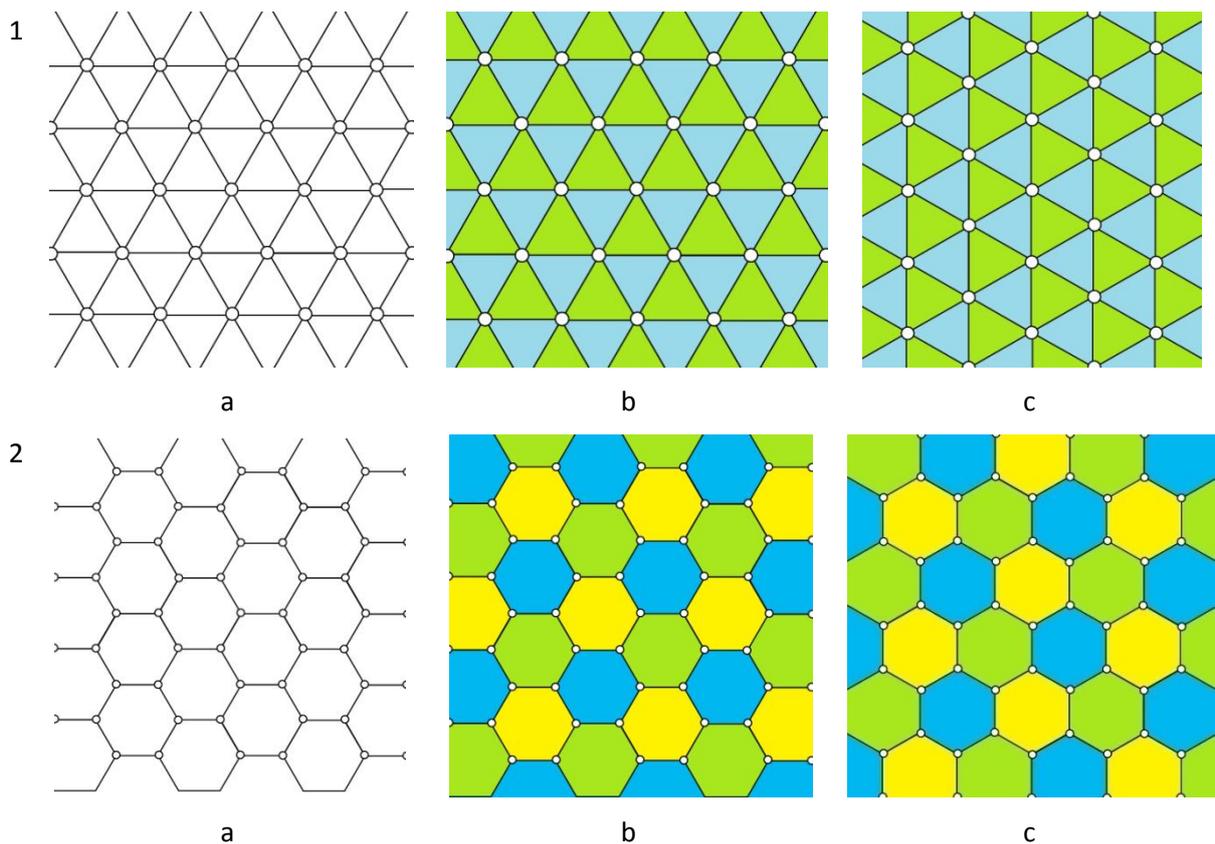
In a case of hexagonal lattice there are two types of lattices also: *hexagonal lattice with triangular tiling* (*Figure 14, 1*) and *hexagonal lattice with honeycomb structure*, which has *hexagonal tiling* (*Figure 14, 2*). Rivet in [46] called those lattices as the *triangular lattice with hexagonal symmetry* and the *hexagonal honeycomb lattice*. The term “hexagonal lattice” is the most frequently in a connection with LGCA modelling and it is used below. Under this term the triangular lattice with hexagonal symmetry is understood.

The hexagonal lattice (triangular lattice with hexagonal symmetry) was chosen for the next development of LGCA models. It consists from equilateral triangles. There are four possible orientations of such triangle. When triangles are pointing up and down, it is *hexagonal lattice with horizontal rows*, as it is shown in *Figure 14, 1 (a) and 1 (b)*. Exactly this type of lattice was used in advanced LGCA model, such FHP models. Second type is a *hexagonal lattice with vertical rows* – it is the lattice with triangles pointing left and right (see *Figure 14, 1 (c)*). The honeycomb lattice has also two orientations, subsequently, they are: the *honeycomb lattice with vertical rows* – every hexagon has two horizontal sides (see *Figure 14, 2 (b)*) and the *honeycomb lattice with horizontal rows* – i.e. every hexagon has two vertical sides (*Figure 14, 2 (c)*). Those two structures differ by an angle of  $30^\circ$ .

It is evident, that lattices, presented in this chapter, have the different coordination number  $b$ . For example, at the square lattice each node is connected with four nearest nodes – the coordination number  $b = 4$ , in a case of honeycomb lattice  $b = 3$ , for hexagonal lattice  $b = 6$ .



**Figure 13:** Types of the square lattice: upright (a) and diagonal one (b) [64]



**Figure 14:** Geometries of 2D hexagonal lattices: 1 – hexagonal lattice with horizontal (a, b) or vertical (c) rows; 2 – hexagonal honeycomb lattice with vertical (a, b) or horizontal (c) rows

The topology and geometry of the lattice is very important for LGCA modelling. The collision between particles, their propagation and the behaviour of the lattice gas at the boundaries are directly dependent on lattice geometry. For example, the HPP model quickly disappeared because of its high anisotropic behaviour. The HPP model lacked rotation

invariance, which is impossible at the square lattice. The more advanced LGCA models have been developed on the hexagonal lattice. Because no node-independent connection of nearest neighbour nodes can be design on the honeycomb lattice – the lattice has different structure at its odd and even rows; the hexagonal honeycomb lattice with triangular symmetry doesn't satisfy the homogeneity principle of LGCA modelling and wasn't used in LGCA models.

More about ordering of the neighbour nodes and properties of the square and hexagonal lattice is presented in next two chapters.

### **2.3.2.2. Neighbourhoods in the square and hexagonal lattice**

From the LGCA point of view, each lattice consists of channels and nodes. Two nodes, spoiled with a channel are considered to be neighbour nodes. As was mentioned in the Chapter 2.2.2 cellular automata updating rules are local by definition. The same property refers to LGCA models (see Chapter 2.4). The state of a given lattice node and states of its vicinity are only required for the acquiring of the local state of the system. According to [49] the spatial region in which the lattice node needs to be searched is called the *neighbourhood*.

For two-dimensional square lattice, the following types of neighbourhoods are often considered (see *Figure 15*):

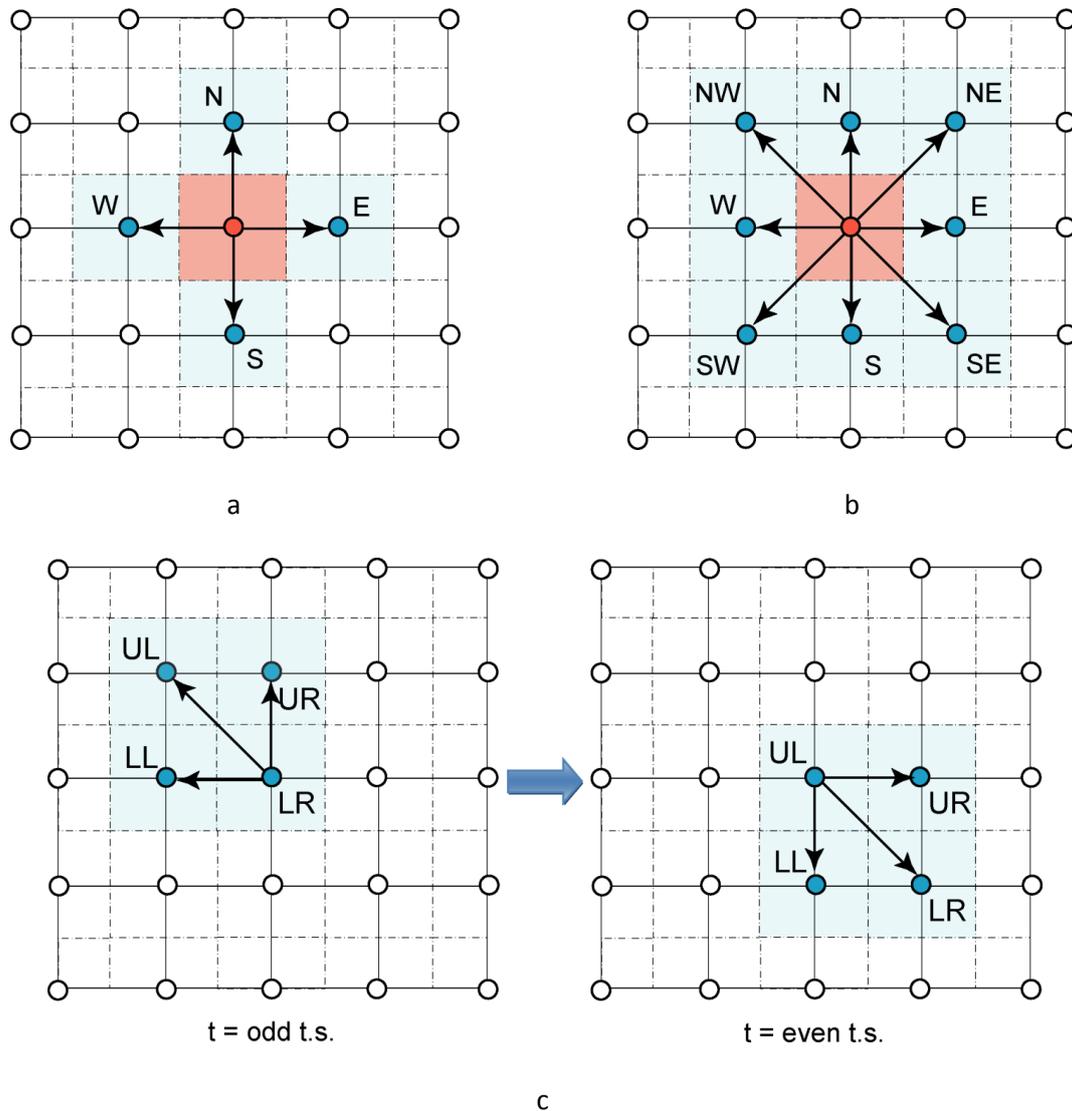
- the von-Neumann neighbourhood;
- the Moore neighbourhood;
- the Margolus neighbourhood.

For a given central node (marked with a red colour in the *Figure 15 (a)*), i.e. the one which is to be updated, the set of the first four nearest neighbour nodes spoiled by the channels with the central node (marked with a blue colour in the same picture), called as a north (N), west (W), south (S) and east (E), creates the *von-Neumann neighbourhood*.

Except those four nodes, the *Moore neighbourhood* contains also second nearest neighbours: north-east (NE), north-west (NW), south-east (SE) and south-west (SW), that is the total of eight nodes – see *Figure 15 (b)*.

In a case of *Margolus neighbourhood* the space is divided into so-called *Margolus blocks* of two-by-two nodes. The definition of nodes inside the block is following: upper-left (UL), upper-right (UR), lower-left (LL) and lower-right (LR). Blocks are shifted by one cell along each dimension on alternate time steps. So, *Margolus blocks* get different spatial coordinates on alternate time steps and nodes inside the blocks. For example, the node labelled as a UR in the following time step (see *Figure 15 (c)*, it is  $t = \text{odd } t. s.$ , i.e. time step) will be become UL at the iteration in the next time step (see  $t = \text{even } t. s.$ , i.e. the right part of the *Figure 15 (c)*). The idea of *Margolus neighbourhood* is that during updating phase, the

transition rule is applied to a whole block at a time rather than a single cell. This principle is being used in block cellular automata or partitioning cellular automaton. [49]



**Figure 15:** Neighbourhood templates for a regular square lattice: von-Neumann neighbourhood (a), Moore neighbourhood (b) and Margolus neighbourhood (c)

Hexagonal lattice is the most favourite one in two-dimensional LGCA models. According to [65] the standard neighbourhood template consists here of six nearest neighbour nodes (in the Figure 16 they are blue). Those cells are edge-connected to the central hexagonal cell (red one in the Figure 16). If the node, located in the centre of the red cell, will be spoiled with central nodes of blue cells using links (i.e. channels), geometry of the hexagonal lattice become to be evident.

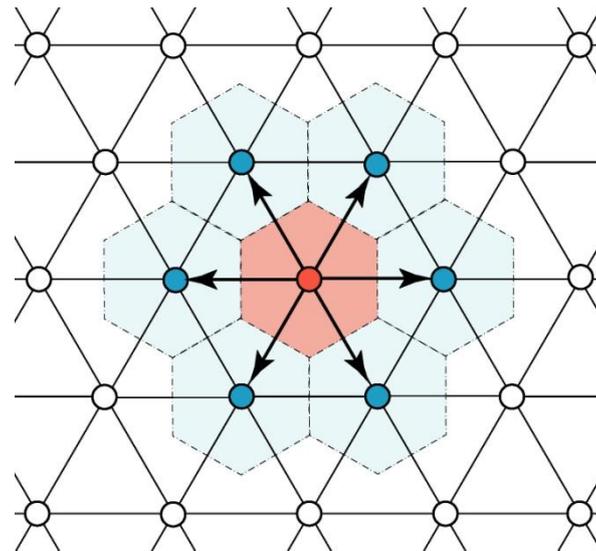


Figure 16: The hexagonal neighbourhood

It is well known, that hexagonal lattice is difficult to represent and to visualize. In simulation algorithms as well as in an image processing the hexagonal lattice is often mapped into the square one. [65] In connection with image processing there are two main techniques how to obtain the hexagonal lattice. One of such techniques is presented in [66]. The *Mersereau's* method is based on a suppression of the alternate rows and columns from the square lattice as shown in the *Figure 17*.

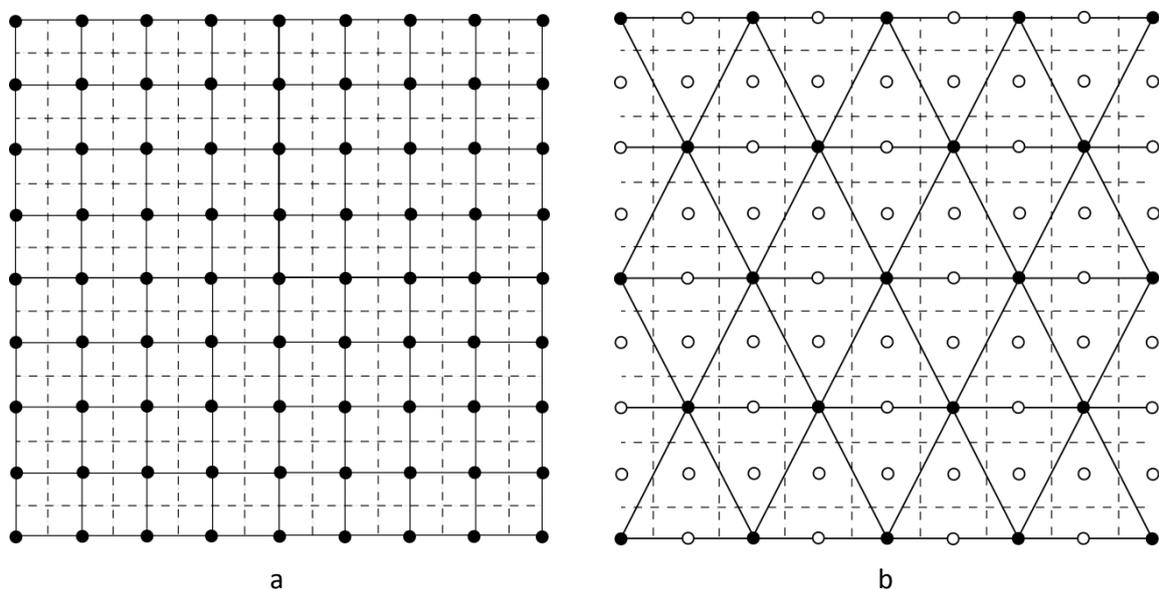
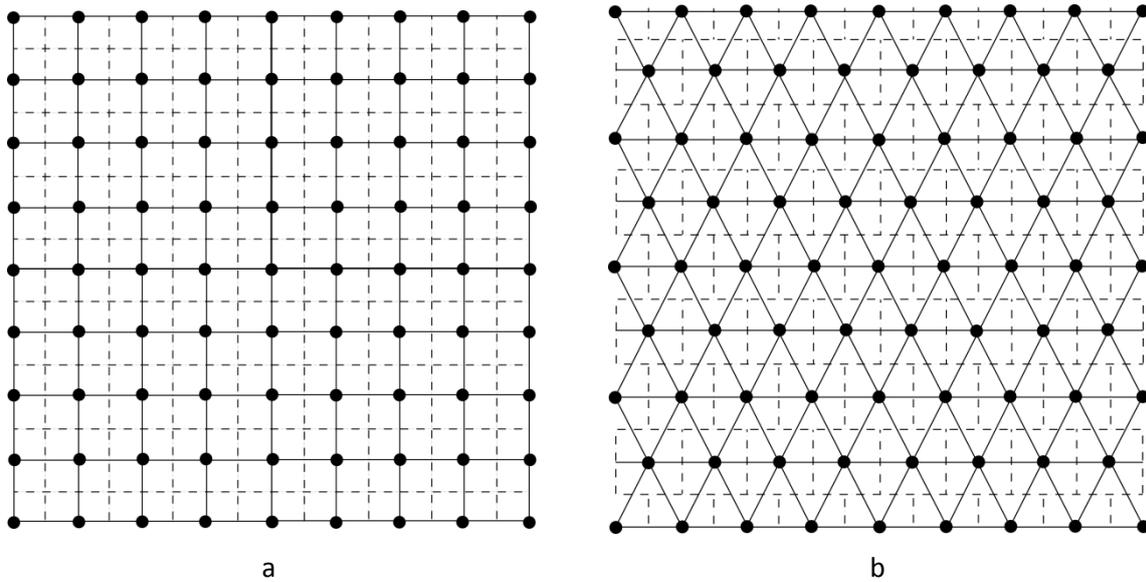


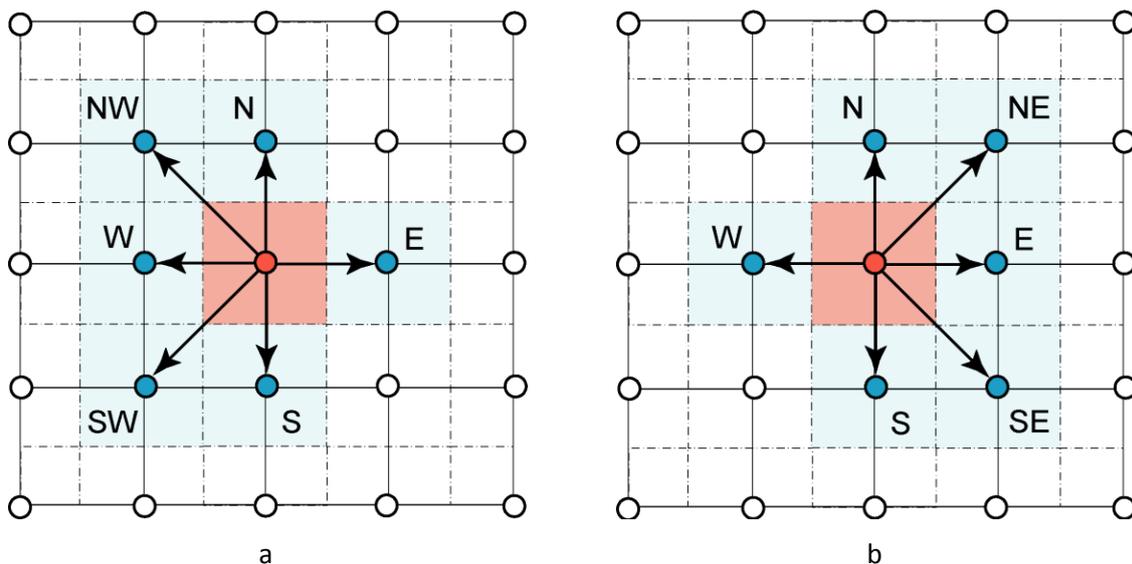
Figure 17: Mersereau's scheme for obtaining the hexagonal lattice (b) from the square one (a)

Another method was proposed by *Staunton* [67]. He has shifted the alternate rows of the square lattice by the half of the pixel's distance (see *Figure 18*).



**Figure 18:** Staunton's method for obtaining hexagonal lattice (b) from the square one (a)

According to *Weimar* [68] in CA simulation the hexagonal type of neighbourhood could be also done through a shift of even rows of the square lattice in one direction and odd rows into the opposite direction. As a result the alternating neighbourhood arises for even and odd rows. So, the neighbourhood in all odd rows contains nodes: N, NW, W, SW, S and E; whereas the neighbourhood in all even rows contains nodes: N, W, S, SE, E and NE (see *Figure 19*).



**Figure 19:** Adaptation of the hexagonal neighbourhood to the square lattice: the ordering of the neighbour nodes in all odd (a) and even (b) rows

### 2.3.2.3. Comparison of square and triangular lattice with hexagonal symmetry

With regards to LGCA modelling, the gas dynamics at the macroscopic level is strongly dependent on the type of underlying lattice on the one side, and type of neighbourhood on the other side. Compared with the triangular lattice with the hexagonal symmetry, square lattice has a simpler representation using square arrays, and also easier visualization. On the other hand the square lattice is jotted in literature as a most anisotropic one from all possible two-dimensional lattices. For example, fluid models based on a square lattice suffer from the preferred directions of the lattice; as a result there are preferred axes for flow propagation. This property is discordant to the physical laws, (in a steady state of the system a certain variable has the same value in all directions, in other words, it is isotropic). That is why the development of HPP model was stopped at a moment, when the anisotropy of its square Bravais lattice broke the isotropy. [69] However, the regular square lattices (upright and diagonal) are usually adopted in other models based on CA because of their simpler computer implementation and computations connected with them. [65]

The hexagonal lattice has been shown in literature as a one, which geometry is suitable for modelling the behaviour of a large class of natural systems. The main ones are hydrodynamic phenomena, diffusion of gasses, crystal growth and so on. This type of lattice has the lowest anisotropy of all regular two-dimensional lattices. The lower anisotropy of the lattice makes simulations more natural, what is very important in lattice gas models for fluid flow. The main disadvantage of the lattice is a difficult representation and visualization. Thus, the mapping of this lattice into the square one it is applied in many simulation algorithms. [70]

## 2.4. Lattice gas cellular automata – principles of the model

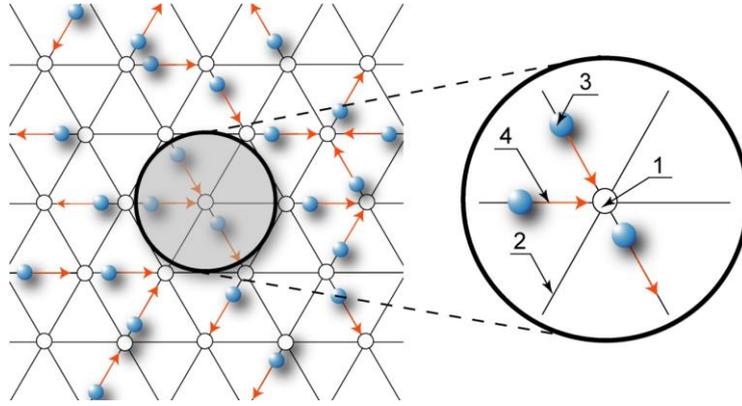
As it was mentioned earlier in [50], the points of view from which a fluid can be described are: microscopic, mesoscopic, and macroscopic. The detailed behaviour of the fluid in the continuum macroscopic level is provided by partial differential equations, e.g., *Navier-Stokes equations* for flow of incompressible fluid (see Chapter 1.4.1). Some other numerical techniques, such as finite-difference and finite-element methods, are being used for the transformation of a continuum system into a discrete one [51]. The lattice gas models based on cellular automata are representatives of fully discrete models. Based on the detailed information about individual particles, such as their positions, masses, and velocities, they enable to describe the behaviour of fluid systems at a molecular level under various macroscopic, microscopic or mesoscopic conditions. Thus, lattice gas models entered into the history as an alternative for fluid system's modelling. Detailed description of lattice gas cellular automata is accessible for example in [31, 46, 49].

According to *Reviet* [46]: “from the mathematical point of view, a lattice gas is a particular class of cellular automata...”. Thus, lattice gas models are based on cellular automata rules and must also satisfy following conditions:

1. *Individual automata* of lattice gas cellular automata are tied geometrically to the nodes of a regular Bravais lattice of dimension  $D$  (as it was mentioned in Chapter 2.2.1, in lattice gas cellular automata the term “individual automaton” is used instead of term “finite automaton”). That is why the individual automaton can also be called a “node” (see *Figure 20, 1*). Nodes are labelled by their position vector  $\mathbf{r}$ , which takes only discrete values. All individual automata are taken to be identical.
2. Any individual automaton has  $2^b$  possible internal states, where  $b$ <sup>16</sup> – it is a Boolean variable, it is integer and it represents the number of *channels* (or *communication channels*) between nodes (see *Figure 20, 2*). Channels are also tied geometrically to the Bravais lattice – in fact, they are links between neighbour nodes of the lattice. In this work channels will label by an integer  $i$  ranging from 1 to  $b$  or 0 to  $b$ . The labelling is node-independent.
3. Similarly to cellular automata, the elementary evolution process of LGCA is repeated at discrete time steps and it is separated by a time increment  $\Delta t$ . In lattice gas models  $\Delta t$  is equal to the unity (time unit – *t. u.*), when the information presented in channel  $i$  at the node  $\mathbf{r}$  goes to the node  $\mathbf{r} + \mathbf{v}_i$ , where  $\mathbf{v}_i$  is the velocity vector (see *Figure 20, 4*). Here the information is presented by fictitious *particles* occupying channels  $i$ . The maximal number of particles in a node is done by  $b$ . In the most part of the lattice gas models (in the non-thermal LGCA: HPP, FHP-1) particles (*Figure 20, 3*) of the same mass  $m$  (in *m. u.* – mass unit) and velocity  $v$  are moving on an underlying regular Bravais lattice, which has the unitary distance  $\Delta l$  (in *l. u.* – length unit) between neighbouring nodes. But there are also multi-speed lattice gas models: FHP-2 and FHP-3 LGCA models that contain extra particles with zero velocity; particles in a GBL model, named after *Grosfils, Boon and Lallemand*, has three different velocities.
4. The elementary evolution process of LGCA is a sequence of two phases: the *collision* and the *propagation* one.

---

<sup>16</sup> In the HPP and FHP-1 lattice gas models  $b$  is a coordination number.



**Figure 20:** Representation of the LGCA model underlaid by the hexagonal Bravais lattice: **1** – the node, i.e. the individual automaton, **2** – the channel, **3** – the moving particle, **4** – the direction of moving [17]

From the information presented below it is obvious that the main characteristic of the LGCA models is a fully discreteness, since main parameters are discrete. Basically LGCA models differ by collision rules. More about collision phase in non-thermal LGCA models is introduced in the following Chapter 2.4.1.

### 2.4.1. Collision phase

In Lattice Gas Cellular Automata the collision phase occurs in each time step, either before or after the propagation phase. The order in the sequence of collision and propagation phases is unimportant when long-time behaviour is considered and large-scale properties are calculated. [46] The collision phase proceeds in accordance with collision rule, which is chosen to conserve a mass (in fact the number of particles) and a momentum at each site of the lattice. The conservation of the local particle number  $n$  and the mass  $m$  at the node  $\mathbf{r}$  is described as follows. Conservation of the particle number at the node  $\mathbf{r}$  is following:

$$\sum_{i=1}^b {}^n n_i(\mathbf{r}) = \sum_{i=1}^b n_i(\mathbf{r}) \quad (28)$$

Mass conservation at node  $\mathbf{r}$  is:

$$\sum_{i=1}^b {}^n m_i(\mathbf{r}) {}^n n_i(\mathbf{r}) = \sum_{i=1}^b m_i(\mathbf{r}) n_i(\mathbf{r}) \quad (29)$$

In Equations (28) and (29) the initial distribution of the colliding particles in the node  $\mathbf{r}$  at individual channels  $i$ 's is represented by  $n_i(\mathbf{r})$ , while their post-collision state in the same node and channel is given by the "new"  ${}^n n_i(\mathbf{r})$  values. It is evident, if the individual masses of all particles in the node  $\mathbf{r}$  are equal to 1, then the total mass in the node  $\mathbf{r}$  is equal to the total number of particles.

The local momentum conservation during the collision phase may be expressed using its components  ${}^n p_a(\mathbf{r})$  and  $p_a(\mathbf{r})$  as:

$$\sum_{i=1}^b {}^n p_i(\mathbf{r}) = \sum_{i=1}^b {}^n m_i(\mathbf{r}) {}^n n_i(\mathbf{r}) v_{ia}(\mathbf{r}) = \sum_{i=1}^b m_i(\mathbf{r}) n_i(\mathbf{r}) v_{ia}(\mathbf{r}) = \sum_{i=1}^b p_i(\mathbf{r}) \quad (30)$$

where  $a = 1, \dots, b$  denotes the components of velocity vector (ne number of velocity components is given by the connection number  $b$ ).

The redistribution of particles in an individual node obeys the rule of keeping the total momentum in the node after the collision phase invariable.

The post-collision state in the node  $\mathbf{r}$  depends only on its pre-collision state and collision rules, which differ for different LGCA models.

### 2.4.1.1. Collision rules of the FHP-1 and FHP-2 LGCA models

Historically, the first lattice gas model was introduced in early 1970's by Hardy, de Pazzis and Pomeau and was called after its authors as a HPP lattice gas model. But the anisotropic properties of lattice gases living on the square two-dimensional lattice were founded. Therefore, the more advanced LGCA models have been developed on two-dimensional hexagonal lattice. The group of so-called FHP LGCA model was introduced ten years later by Frisch, Hasslacher and Pomeau. Several versions of FHP model have been developed with the same geometrical lattice structure having different collision rules. This group of models is described in details by Rivet in [46] and is introduced in this study.

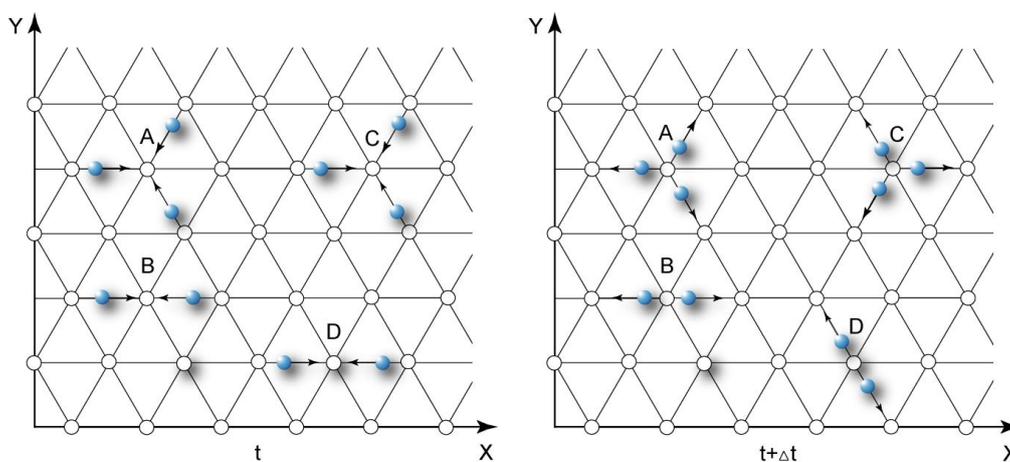
#### *FHP-1 lattice gas cellular automata model*

An individual automaton of the FHP-1 LGCA model has  $b = 6$  channels, corresponding to the six directions of the hexagonal lattice. Channels are labelled by  $i = 1 \dots 6$ . The masses  $m_i$  of all particles are equal. Usually,  $m_i = m = 1$ . The absolute value of momentum  $p_i$  of each particle is numerically equal to  $v_i$ , this is physically consistent with unit mass and unit time step. [46] Unit time step in LGCA models is a period which particle needs to jump from the certain node to neighbour one.

During the collision phase in the FHP-1 model two-particle and maximum six-particle collisions may occur.

#### *Two-particle collisions in the FHP-1 LGCA*

If two particles meet together in a same lattice node, they yield two-particle collision. An example, when particles come from opposite directions is presented in *Figure 21*, nodes B and D. With equal probabilities particles are being rotated by  $+60^\circ$  or  $-60^\circ$  (*Figure 21*, node D), or continue in a same direction of moving.



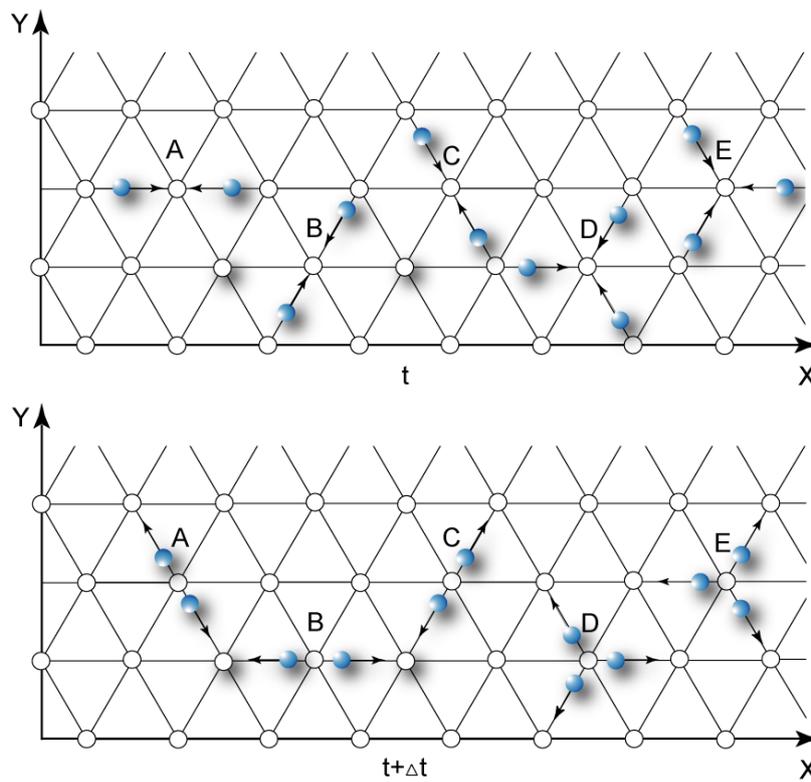
**Figure 21:** Typical two- and three-particle collisions in the FHP-1 LGCA model [17]

*Three particle collisions in the FHP-1 LGCA*

When three particles meet simultaneously in one node (see *Figure 21*, nodes A and C), it is three-particle collision. Collision either takes place with a rotatory deflection of the velocity vectors by  $60^\circ$  (*Figure 21*, node A) or their orientation remain unchanged (*Figure 21*, node C). The rotation by  $-60^\circ$  leads to an identical local state transition.

*Effective collisions in the FHP-1 LGCA*

In the FHP-1 LGCA model there are  $\sum S = 2^b = 2^6 = 64$  various local states (i.e. states of the individual automaton). Among all these states five of them are effective. Effective state is a result of effective collision. The collision is considered to be effective, when the rotation of the velocity vectors by  $60^\circ$  has a place. All effective collisions of the FHP-1 LGCA model are presented in the *Figure 22*. Three of them are two-particle collisions (*Figure 22*, nodes A, B, C), and two of them are three-particle ones (*Figure 22*, node D and E).



**Figure 22:** *Effective collisions in the FHP-1 LGCA model [17]*

The complete review of all possible pre- and post-collision local states in FHP-1 LGCA model is introduced in *Appendix A*. For a coding and interpretation of the pre- and post-collision states the binary or decimal systems are using in some LGCA algorithms. It was taken in that appendix into account.

***FHP-2 lattice gas cellular automata model***

The FHP-2 LGCA model is a variant of the FHP-1 LGCA model that includes the possibility of one rest particle per node in addition to the six moving particles of FHP-1 model. An individual automaton in FHP-2 model has  $b = 7$  channels, corresponding to the six directions of the hexagonal lattice (i.e. six moving particles can be found in a lattice node) and one place for a rest particle. The channels corresponding to moving particles are labelled by  $i = 1..6$ , and the channel corresponding to the rest particle is labelled as  $i = 0$ . The masses  $m_i$  of all particles are equal to one and absolute value of the momentum  $p_i$  of each particle is equal to  $v_i$  except the momentum of the rest particle which is zero. [46]

The collision rules of the FHP-2 LGCA model are the same as at the FHP-1 model with two additional events. A moving particle arriving at a node with a rest particle produces a pair of moving particles at angles  $+60^\circ$  and  $-60^\circ$ , regarding the direction of the incoming particle (see *Figure 23*, G). The last additional collision event is the reverse to the former. Two colliding particles in a node with their velocity vectors at  $120^\circ$  angle result in one resting particle and in one moving particle moving in the direction of their original pre-collision momentum vector (see *Figure 23*, H). In the FHP-2 LGCA model there are  $\sum S = 2^b = 2^7 = 128$  various local states, twenty two of them are effective ones. Thanks to the effective collisions with resting particles, the FHP-2 model doesn't conserve kinetic energy. It is assumed that either the energy is exchanged with an adjacent thermodynamic reservoir or the resting particles vibrate with a vibrational energy equalling their original kinetic one.

The examples of several collisions of the FHP-2 LGCA model are presented in *Figure 23*. Collisions in nodes A, B and E are similar to the two and three-particle collisions, presented in a description of the FHP-1 LGCA model. The two-particle collision between one moving and one rest particle is shown in the node G. Examples of the three-particle collisions, where one of the particles is a rest one, are presented in *Figure 23* in the nodes C and D. In the node H two-particle collision is illustrated. In contrast to the FHP-1 LGCA model, it results stopping one of the moving particles. This type of collision is opposite to the example in the node G.

Effective collisions of FHP-2 LGCA model are depicted in nodes A, B, C, D, G and H. All possible pre- and post-collision local states of the FHP-2 LGCA model are introduced in *Appendix B*.

Collision rules in LGCA models can be deterministic but is more often they are non-deterministic (probabilistic). In *Figure 23* nodes C and D have the same pre-collision state. According to the information presented in *Appendix B* there are three possible post-collision states. Two of them are depicted at the right part of the picture (*Figure 23*) in nodes C and D. The third one is a collision without rotary deflection, thus is non-effective one. In deterministic LGCA model the post-collision state is always pre-defined. If LGCA model is the

probabilistic one, than the post-collision state of the node is probabilistically chosen between possible states according to collision rules.

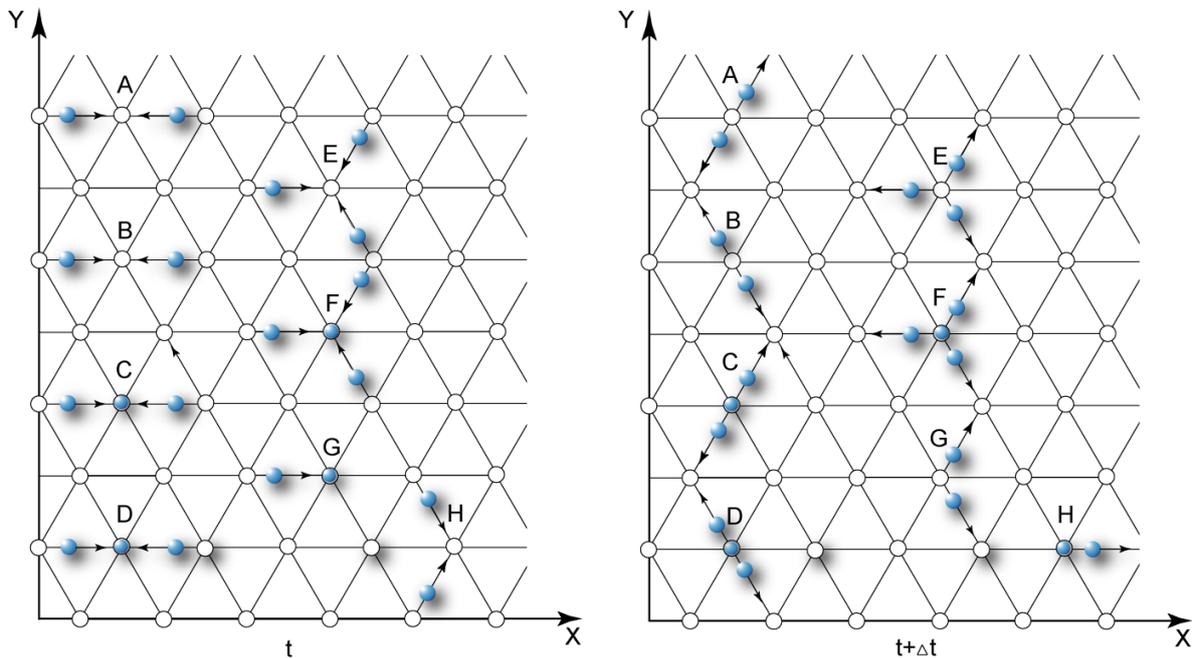


Figure 23: Two- and three-particle collisions in FHP-2 LGCA model [17]

After the collision phase, the newly acquired information propagates from the node to neighbour nodes – thus the propagation phase occurs.

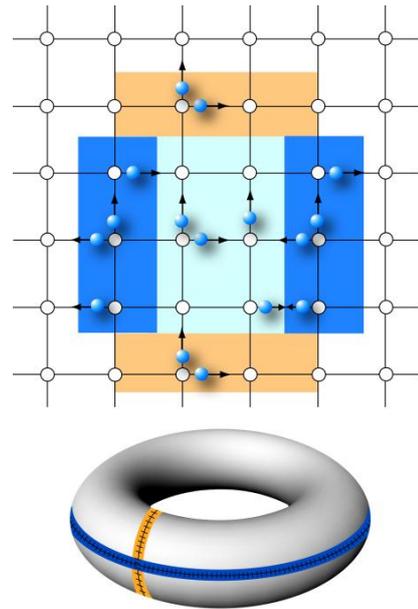
### 2.4.2. Propagation phase in LGCA models

During the propagation phase, a particle is shifted from the node  $\mathbf{r}$  to the node  $\mathbf{r} + \mathbf{v}_i \Delta t$ , i.e. if a particle is present at a moment  $t$  in a node  $\mathbf{r}$ , it is shifted to the neighbouring node at time  $t + \Delta t$  according to the direction  $i$  of velocity vector. This type of propagation phase takes a place inside the whole Bravais lattice. At the boundaries of the lattice there are various methods how to realize the propagation of particles.

One of the methods is so-called “*periodic boundary condition*” [46]. In that case the boundary parts of the lattice, on which the propagation phase is implemented, has to be connected to the form of a loop (see Figure 24). This wrapping of opposite sides of a finite lattice leads to a periodic motion of the individual particles. The escaping particles return to the finite lattice on the opposite sides of its boundaries.

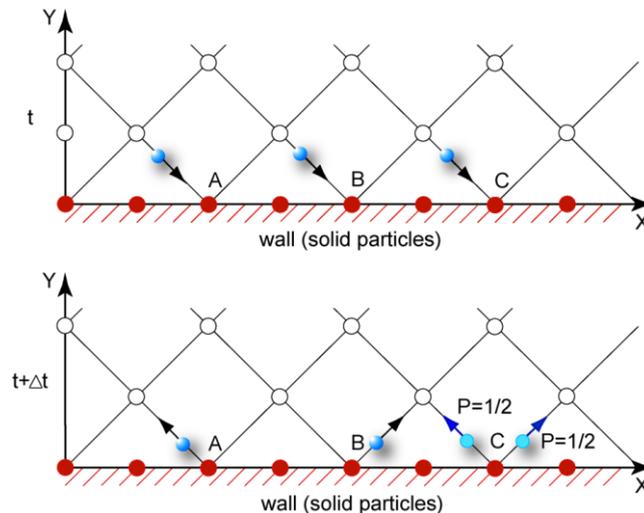
Another method is in conflict between the theoretically infinite lattices used in LGCA models and limited memories of computers. This method is called as a *reflective boundary condition*. This type of boundary conditions is based on various types of particle collision with solid walls (see Figure 25). According to Rivet [46] the following types of reflections are:

1. *Bounce-back reflection* – also known as a *no-slip boundary condition*. When a particle reaches the wall, its momentum vector is changed with central symmetry, thus the particle is being sent back in a same direction to where it comes from (see *Figure 25*, node A).
2. *Specular reflection* – is also known as a *free-slip boundary condition*. The vector component of particle momentum, parallel to the wall surface, is conserved during such a collision, while the normal component of it is reversed (see *Figure 25*, node B).
3. *Diffusive reflection* – is a combination of the bounce-back and specular reflections, it is occurring with chosen probabilities  $P$  (see *Figure 25*, node C).



**Figure 24:** The principle of periodic boundary conditions for two-dimensional square LGCA (HPP model [17])

Red nodes in the *Figure 25* represent moveless particles of the solid surface (for example, solid walls or surface of any obstacle). Black arrows illustrate the momentum vector of particles. Blue arrows show the possible directions of the momentum vector as a result of diffusive type of reflective boundary condition.



**Figure 25:** Various reflective boundary conditions: **A** - bounce-back reflection, **B** - specular reflection, **C** - diffusive reflection [17]

### 3. BASIC ALGORITHM BASED ON THE FHP-1 LATTICE GAS CELLULAR AUTOMATA

*“The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work.”*

*John von Neumann*

The creation of the own Lattice Gas Cellular Automata algorithms based on the FHP-1 LGCA model is presented in this chapter. The algorithm is proposed for fluid flow simulation. Main blocks of the basic LGCA algorithm developed for general-purpose computers, their specification and function within the whole algorithm are analysed in detail. Modifications of the basic algorithm depending on problems under investigation are presented later in Chapters 4-6.

A large variety of computers from personal computers to powerful parallel supercomputer and a wide range of programming languages explain the existence of a quantum of lattice gas algorithms, which have been implemented since 1985. The algorithm developed here for fluid flow modelling was briefly described in [17] and is explain in detail in this work. Structure of the algorithm includes unchangeable part that can be used as a base for each new algorithm independent on the concrete choice of a lattice gas model (collision and propagation phases in the concrete).

The algorithm was created in a C++ programming language Borland version 4.0 and its full text is presented in *Appendix C*. Algorithm was formally divided into ten code fragments. This structure is conserved in all algorithms created in the frame of this work. Follow description explains only the role and the function of certain algorithm code fragments. My own concept of the FHP-1 LGCA from technical point of view is noticeable from the *Appendix C*.

#### 3.1. Code fragment 1 – Header files and initialization of the simulation domain

Usually, in computer programming and particularly in the C++ programming language, header files stands at the beginning of the algorithm. Header files commonly contain the following:

- definition of standard library functions;

- declaration<sup>17</sup> of variables;
- declaration of subroutines and other identifiers.

Consequently, the own algorithm starts with an enumeration of all standard library functions, which will be used during the calculations. In our case these standard libraries are:

*graphics.h* – is a definition and declaration for graphical library [72].

*stdlib.h* – Standard General Utilities Library includes dynamic memory management, random number generation, integer arithmetics, etc.

*stdio.h* – Library to perform Input/Output operations [73].

*conio.h* – used in MS-DOS compilers is being used to create text user interfaces, it is not describes in The C Programming Language book or in the C standard library [72].

*math.h* – Numerics Library declares a set of function to compute mathematical operations and transformations.

*float.h* – describes the characteristics of floating-point types.

*time.h* – Time Library contains definition of functions to get and manipulate date and time information [73].

In the next step the initialization<sup>18</sup> of the simulation domain, the space where the computer simulation takes place, is made. The biggest size of the adjacent domain is chosen the higher accuracy of the outcomes is expected. Regarding to the computer power maximum 450 single points (lattice nodes) in the direction *OX* (*DIRX* 450) and 300 points in the direction *OY* (*DIRY* 300) were chosen, where *OX* and *OY* are *x* and *y* axes of the Cartesian system of coordinates.

In a part named “Variables declaration” all variables, which will be used in a main part of the algorithm, are enumerated. Type of variables (*int* – integer type or *float* – floating point type) and their identifiers are declared. Based on a size of the simulation domain identifiers *xmax* and *ymax* gets values 449 and 299 lattice nodes accordingly to *DIRX* and *DIRY*. Algorithm is working with a big amount of information. In each time step one needs to know which channels in the concrete node are occupied, what is the number of particles (parameter *mass*) and their total velocity. Twelve different arrays<sup>19</sup> were declared for that reason:

- *vx* and *vy* – contain information about *x* and *y* components of a total particle velocity in the particular lattice node;

---

<sup>17</sup> *Declaration* specifies identifiers – the single objects in C++ language. Declaration of variables contains specification of type and other aspects [72].

<sup>18</sup> *Initialization* is an assignment of a value to the declared variable [72].

<sup>19</sup> *Array* is a series of elements of the same name and type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier and can be used independently on each other [92].

- $nv_x$  and  $nv_y$  – are using during the calculation of a new particle velocity distribution in a particular lattice node during the implementation of propagation phase;
- $m$  and  $nm$  are arrays, where the instantaneous number of particles in the particular node and the new number of them after collision and propagation phases are recorded;
- $i_1, i_2, i_3, i_4, i_5, i_6$  – contains information about channel occupation.

Probability of a channel occupation is given by the special parameter  $pco$ , thus the density of simulated fluid is treated. Because the system evolves in time, parameters  $cycle$ ,  $cmax$  and  $series$  are included, where total number of time steps is given by  $cmax$ . Variables  $sign$  and  $char\ str$  declare graphic pre-set; parameter  $print$  adjusts the colour of graphic outputs.

Some calculations are performed in subroutines<sup>20</sup>. Three subroutines are declared in the algorithm. The collision and propagation phases are supplying in those subprograms:

```
int collision(void);
float propagationodd(void);
float propagationeven(void);
```

Due to the lattice geometry and different ordering of neighborhoods propagation phase takes place in odd and even rows of the lattice separately. Detailed description of these subroutines, their function is described in Chapter 3.6.

### 3.2. Code fragment 2 – Graphic output setting

The main part of the program begins with the setting of graphic outputs. It is a standard part of the algorithm and it doesn't change in the algorithms proposed later in the thesis.

### 3.3. Code fragment 3 – Creation of the simulation domain and initial state of the simulated system

The values of the data-fields (declared arrays) are reset to 0 at the beginning of the algorithm. This operation is called as a "Data arrays resetting".

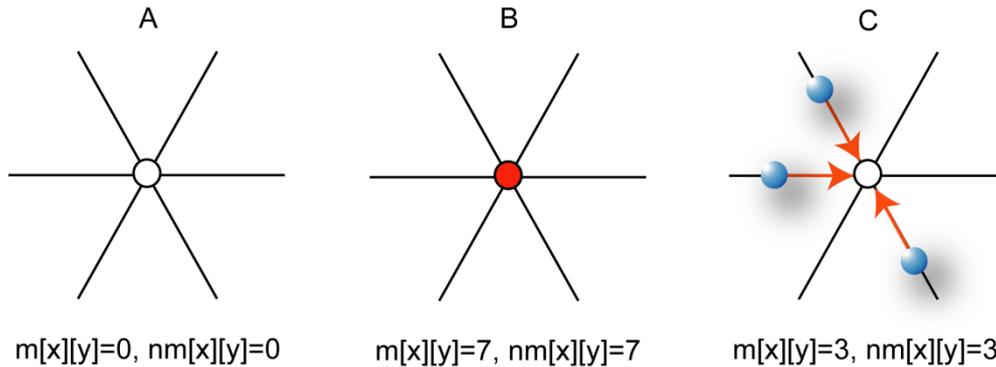
If the simulation model has solid objects as walls of a channel or a cavity, a porous medium etc., a creation of those objects becomes as a first. At the basic simulation model the simulation domain of the size  $450 \times 300$  lattice nodes was created. Fluid particles are usually moving and interacting with each other inside the simulation domain. The domain is confined by solid boundaries. In this algorithm fluid particles are colliding with the solid walls according to the bounce-back type of the boundary reflections.

To distinguish different types of particles I used here several codes. These codes were being related to arrays named as  $m$  and  $nm$  (arrays using for calculation of an instantaneous

---

<sup>20</sup> *Subroutine* (also function, method, procedure, subprogram) is a set of codes, which performs a specific task and can be relatively independent of the main program.

number of particles, i.e. their total mass, in nodes). At every position with coordinates  $x$  and  $y$  the value of parameters  $m$  and  $nm$  varies from 0 to 7. If the  $m[x][y] = 0$  (also  $nm[x][y] = 0$ ), i.e. if the node at the position  $[x][y]$  has no particle its mass is 0 (Figure 26, A). If the  $m[x][y] = 7$  (also  $nm[x][y] = 7$ ) the node at the position  $[x][y]$  is occupied by a moveless particle (Figure 26, B). When the  $m[x][y] = 1 \dots 6$  (also  $nm[x][y] = 1 \dots 6$ ), then there is 1 to 6 moving particles at the position  $[x][y]$  of the simulation field (see Figure 26, C). It also means that one to six channels of the node are occupied by a fluid particle, and therefore the total mass at the node is between 1 and 6.



**Figure 26:** Various examples of node's occupation: **A** – an empty node, **B** – node occupied by a solid particle, **C** – the node occupied by fluid particles

Thus, this part of the algorithm includes creation of the moveless particles at the simulation domain. Detailed description of a principle, which was implemented during occupation of channels by moving particles, is described in Chapter 3.4.

### 3.4. Code fragment 4 – Occupation of channels by fluid particles

#### 3.4.1. Geometry of the lattice

According to the Rivet's definition [46] individual automata of Lattice Gas Cellular Automata are being geometrically tied to the nodes of the regular Bravais lattice, embedded in a D-dimensional Euclidean space. Square and hexagonal types of lattice are being used in two-dimensional Lattice Gas Cellular Automata models.

As it was explained in the Chapter 2.3.2, Bravais square lattice offers simple representation and visualization. It uses square arrays, but simulation results show anisotropic behaviour of LGCA. This is unfit for modelling of physical phenomena. Therefore the square Bravais lattice wasn't used for the LGCA algorithm in this work.

The hexagonal honeycomb lattice was illustrated in the Chapter 2.3.2.1. Every structural element of the lattice has a small number of neighbours (only three) that could be useful in some cases. Representation and visualization of that lattice are more difficult as for square

one because it must be mapped to square arrays and display lattice nodes. Hexagonal honeycomb lattice is a regular but not a Bravais one. This type of lattice also wasn't used, because of the small number of neighbours.

The hexagonal lattice has the lower anisotropy compared with the square lattice; thus the simulation systems appear more natural and correct. The greatest disadvantage of the lattice is more difficult representation and visualization.

Based on conclusions of the Chapter 2.3.2.2 the hexagonal lattice was created at the regular square lattice by the relative shifting of odd and even rows with each other. Position of channels  $i_1, i_2, i_3, i_4, i_5, i_6$ , which connect any lattice node with neighbour lattice nodes, is partially different and depends whether the node inheres in odd or even row of the lattice (see Figure 27). Ordering of channels in odd and even rows is evident from the Figure 27 (b). Position of the neighbour nodes connected to the node  $(x, y)$  is separately presented for odd and even rows in the Figure 28 (a) and (b). Blue colour is used for labelling the channels related to nodes in odd rows of the lattice, the red one – in even rows.

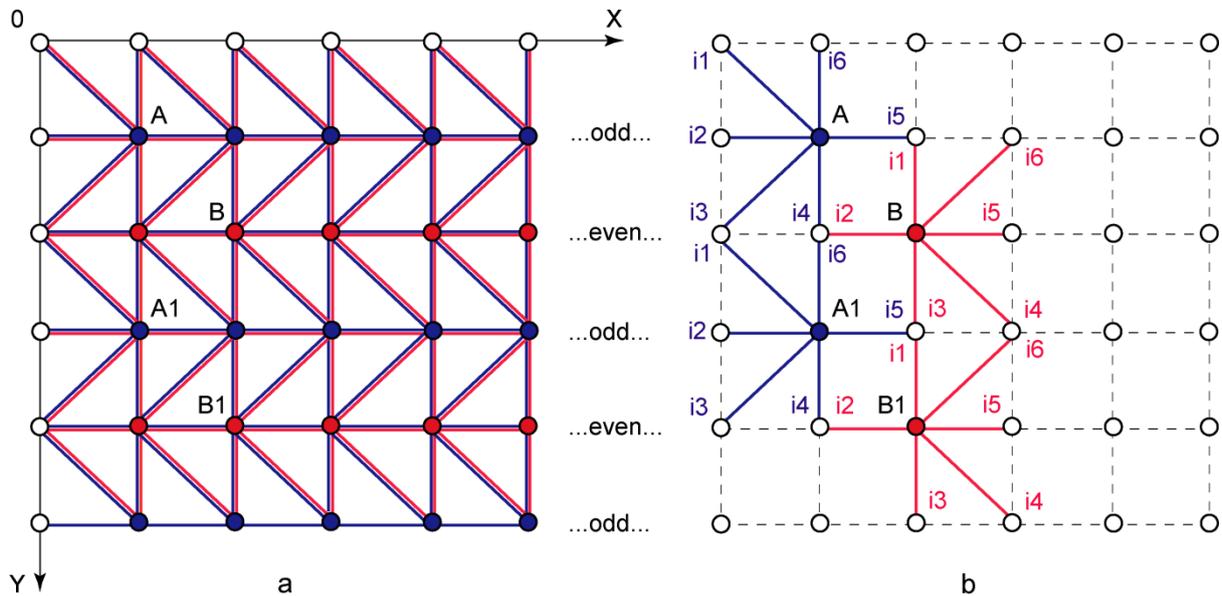
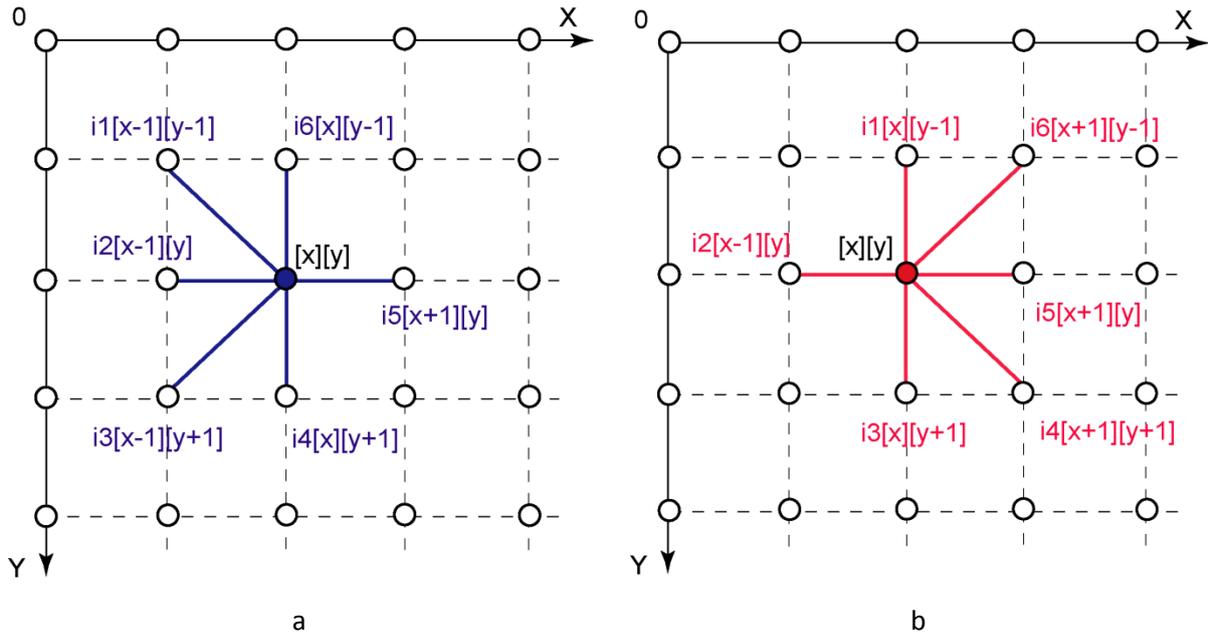


Figure 27: The hexagonal lattice as the equivalent square lattice with an additional diagonal connection (a) and the regular hexagonal neighbourhood in odd and even rows of the lattice (b)

Based on Figures 27 and 28 it can be acquired the false impression, that the distance to diagonal neighbours is longer than to nearest ones and that they are not equiangular. But according to the geometry of hexagonal lattice all distances are equal. Thus the value of the distance between every pair of neighbour nodes is taken to be 1 lattice unit (l. u.) and the angle between neighbour channels is taken to be  $60^\circ$ . That assumption is valid for all implemented calculations in the algorithm.



**Figure 28:** The ordering of the channels  $i1 \dots i6$  and the determination of the neighbour nodes position in all odd (a) and even (b) rows of the lattice

Because odd and even rows were differentiated in the algorithm, some operations were implemented separately for odd and even rows in a case, when the position of channels was important.

### 3.4.2. Occupation of channels by fluid particles

In this part of the algorithm an initial state of the simulated system is generated. As it was mentioned in theoretical part of the thesis, the state of the LGCA is given by states of its individual automata (see Chapter 2.4). The state of individual automaton is generated by means of the channel occupation by moving type of particles.

Thus, generation of moving particles takes place on resting empty nodes of the simulation domain, where no moveless particles are taken place. This process is fully random and is driven by the probability of channel occupation, called as a  $pc_o$  (see Code fragment 1 – i.e. Chapter 3.1). Each channel in every lattice node randomly takes the value 0 or 1 according to the following steps:

1. Selecting the lattice node with coordinates  $x$  and  $y$ ;
2. Detecting the information about the number of particles  $m$  in the lattice node. If the node is occupied by a moveless particle (the mass  $m$  in the node got value 7), return to the step 1;
3. Selecting a channel of the lattice node and checking the possibility of its occupation by fluid particle. The number of particles in the neighbouring node connected with the selected channel should be less than 6; the occupation of the channel by moving particle is then possible;

4. Generation the random integer number in the range of 0 to  $pc0$ , where  $pc0$  is the probability of channel occupation. In the algorithm the parameter  $pc0$  can range from 2 to  $\infty$ <sup>21</sup>. The highest value the parameter  $pc0$  has, the lowest probability of channel occupation is. If the random number is equal to 1, the fluid particle is situated on the channel; channel takes value 1 and parameter  $m$  in the node at the position  $(x, y)$  increases by 1. In other cases, when the channel of the lattice node remains unoccupied, algorithm does return to the step 3. As an example, occupation of the channel  $i1$  at every lattice node in odd rows is:

```

if (m[x-1][y-1]<7) {I1=random(pc0);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

```

5. Repeating steps 3 and 4 for all the lattice node channels;
6. Calculating the  $x$  component of the total particle velocity in the lattice node  $vx$  with respect to channel occupation and a fact that directions of velocity vectors of moving particles are inward the node;
7. Calculating the  $y$  component of the total particle velocity in the lattice node  $vy$  with respect to channels occupation and a fact that directions of velocity vectors of moving particles are inward the node;
8. Repeating the whole procedure for all lattice nodes with regards of odd and even rows of the lattice and related channels location.

### **3.5. Code fragment 5 – Graphical outputs of the initial system configuration**

If the simulation model has graphical outputs the first graphical image, depicting the initial system configuration, takes a place. According to a number of moving particles in every lattice node (the value of parameter  $m$  in fact), different colours are assigned to different number of particles inside lattice nodes (see *Figure 29*).

Form the *Figure 29* it is obvious that the most dense lattice gas is created by means of the  $pc0 = 2$  (see *Figure 29 (a)*). There are nodes with one, two, three and four moving particles, i.e. one to four channels are randomly occupied. When the value of  $pc0$  is 2000, only two moving particles are randomly generated (see *Figure 29 (d)*).

### **3.6. Code fragment 6 – The main cycle of the algorithm**

The cyclic part of the algorithm consists of collision and propagation phases mainly. These phases repeat subsequently till the variable *cycle* gets the value *cmax* initialized in a header part of the algorithm. The main cycle is given in the algorithm by an expression:

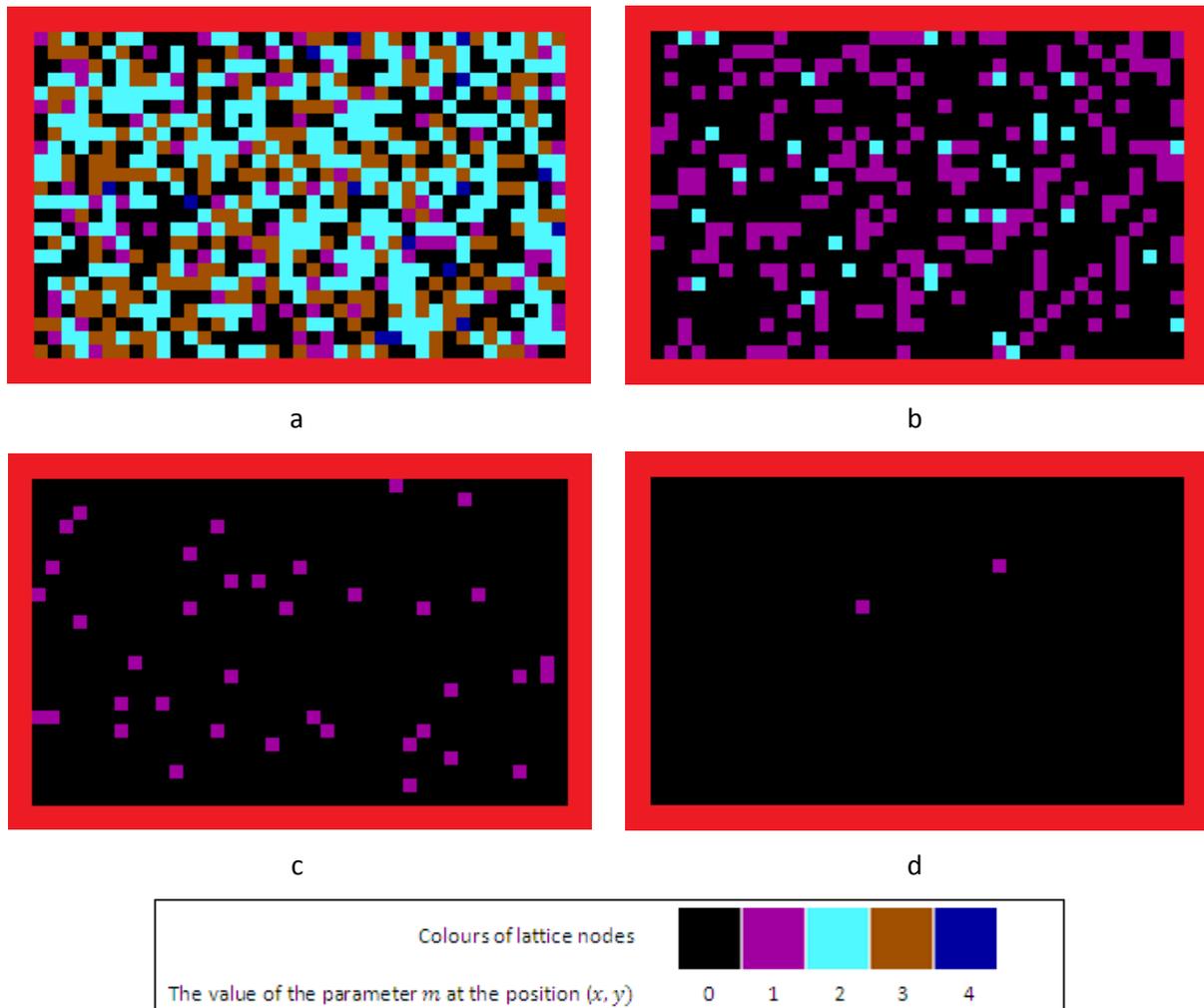
---

<sup>21</sup> When  $pc0 = 2$ , the macro *random (pc0)* returns a random number in the range 0 to 1. Thus the probability of channel's occupation is 0,5 – i.e. the average density of fluid moving particles is 3 particles per node.

```
for (cycle=0; cycle<cmax+1; cycle++)
```

The structure of the main cycle is presented by means of flowchart (see *Figure 30*). It is obvious, that during every time step algorithm goes through all nodes of the lattice. When the lattice node is empty (the value of  $m = 0$ ), nothing is happend. When one o more moving particles are located in the node, algorithm aplies collision and propagation phases.

Detailed description of these phases is presented in the next Chapter 3.6.1 and 3.6.2.



**Figure 29:** The initial configurations of the system according to the value of the parameter  $pco$ :  $pco = 2$  (**a**);  $pco = 20$  (**b**);  $pco = 200$  (**c**) and  $pco = 2000$  (**d**).

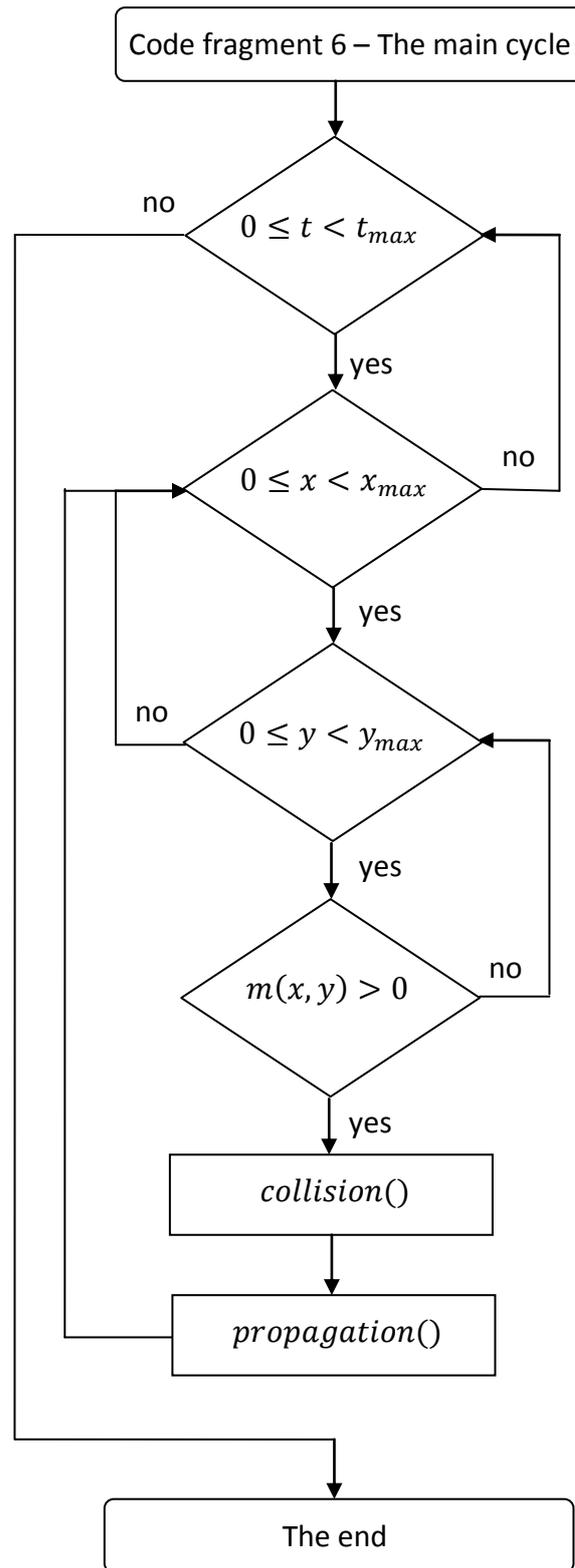


Figure 30: The flowchart representing the main cycle of the developed FHP-1 LGCA algorithm

### 3.6.1. Code fragment 6-A – Collision phase

The collision phase processes homogeneously in all lattice node expecting nodes which being settled by the moveless particle (i.e. nodes which belong to the channel walls or solid obstacle). During this operation the distinction of odd and even rows of the lattice is not needed. The subroutine *collision()* attends to collision phase implementation and consists of following steps:

1. Selecting the lattice node with coordinates  $x$  and  $y$ .
2. Declaration of the subroutine's variables: *cannel*, *mas*, *velx* and *vely*. The variable *mas* is kept equal to the instantaneous value of number of particles  $m$  in the lattice node at the position  $(x, y)$ ; *velx* – is kept equal to the instantaneous value of  $x$ -component of total particles velocity  $vx$  in the lattice node with coordinates  $(x, y)$ ; *vely* – is kept equal to the instantaneous value of  $y$ -component of total particles velocity  $vy$  in the lattice node with coordinates  $(x, y)$ . The values of parameters *mas*,  $vx$  and  $vy$  are the input information for the subroutine *collision()*.
3. Generation of a random number between 0 to 6, where the upper value is given by the number of channels in the lattice node<sup>22</sup>:

`random(6) .`

That macro in FHP-2 lattice gas model returns the random number in the range of 0 to 6 (i.e. *random(7)*)<sup>23</sup>.

4. Choosing the channel at random regarding to the value generated in a step 3. The value  $n$  is generated, the channel  $n + 1$  is being chosen. For example, if the parameter *channel* gets value 0, the channel *i1* is active for a next operation (the adjustment is determined by the numbering of channel that begins with the number 1). If the chosen channel is empty than occupy the selected channel of the node with the particle, i.e. with value 1, and reduce the parameter *mas* by 1:

```

cannel=random(6);
if (cannel==0)
{if (i1[x][y]==1) {goto nav1;}
i1[x][y]=1; mas=mas-1;}
    
```

In a case that the channel is settled by moving particle, go to the step 3.

5. Repeating steps 3 and 4 as long as the parameter *mas* is equal to zero.
6. Calculation the  $x$ -component of the new total particle velocity *velx* of the newly proposed configuration in the lattice node. If the *velx* in this node is not equal to the original input value *velx* (i.e. the difference between newly calculated *velx* and original *velx* is not equal to zero), go back to the step 3.

<sup>22</sup> In fact, "*random(6)*" returns a random number in the range of 0 to 5 (0, 1, 2, 3, 4, 5) – i.e. a random number from the interval [0; 6).

<sup>23</sup> An individual automaton in FHP-2 model has 7 channels, corresponding to the six directions of the triangular lattice with hexagonal symmetry (i.e. moving particles) and to the one place for a rest particle.

7. Calculation the  $y$ -component of the total particle velocity  $vely$  of the newly created configuration in the lattice node. If the  $vely$  in the node is not equal to the original input value  $velx$  (i.e. the difference between newly calculated  $vely$  and original  $vely$  is not equal to zero), go back to the step 3.
8. Registration the information about newly created configuration, i.e. the occupation of individual channels in the lattice node  $i1, i2, i3, i4, i5, i6$  – it is the output information of subroutine *collision()*.
9. Repeating previous steps for all lattice nodes systematically.

### **3.6.2. Code fragment 6-B – Propagation phase**

The propagation phase comes after the collision phase. Because the position of individual channels is important in this part of the algorithm, the propagation phase is implemented separately in odd and even rows of the lattice. The subroutines *propagationodd()* and *propagationeven()* serve for that and contain follow steps:

1. Selecting the lattice node with coordinates  $x$  and  $y$ .
2. Detecting the input information of the lattice node. If the selected lattice node is occupied by the solid moveless particle (i.e. the mass is equal to 7), return to the step 1. The greatest interest at that moment is the occupation of individual channels.
3. Coming through channels of the lattice node and subsequently looking for the first occupied channel denoted in the algorithm as  $i$ . If all channels are empty, go back to the step 1.
4. If the channel  $i$  is occupied, detecting the state of the neighbor node, which communicates with the selected lattice node through the channel  $i$ .
5. In the case that the neighbour node is not occupied by solid particle, relocation the particle sitting in the channel  $i$  to the neighbouring node takes place. The new particle number (the value of the parameter  $nm$ ) in the neighbouring node extends by 1. New values of  $x$ -component  $nvx$  and  $y$ -component  $nvx$  of the total particles velocity in the neighbour node are extended by the value of the  $x$ - and  $y$ -component of the velocity of particle coming through the channel  $i$ .
6. In the case that the neighbouring node, communicating with the selected lattice node through the channel  $i$ , is occupied by solid moveless particle, implement reflection depending on the chosen type of boundary conditions. In the basic algorithm the bounce-back type of particles reflection was used. In that case the new number of particles (the value of the parameter  $nm$ ) in the chosen lattice node extends by 1. New values of  $x$ -component  $nvx$  and  $y$ -component  $nvx$  of the total particles velocity in the lattice node are extended by the values of  $x$ - and  $y$ -component of the velocity of particle coming inside the selected lattice node through the channel  $i$ . In other words instead of displacement the particle from the channel  $i$

to the neighboring node, we move it back to the selected lattice node and all information connected with this particle.

As an example, implementation of steps 4-6, when the channel  $i1$  was denoted as an occupied is presented here:

```

if (i1[x][y]==1)
{
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-
    0.5;
    nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
}

```

7. Repeating steps 5 and 6 for all occupied individual channels of the selected lattice node.
8. Repeating previous steps until all lattice nodes are not visited.

### **3.7. Code fragment 7 – Recording of the new system’s state**

According to Wolfram [74], models based on cellular automata rules are always defined to use the old values of neighbours in order to determine the new value of any particular cell. The C++ program explicitly updates values of lattice gas cellular automata from one side of the simulation domain to other one. As a result, it is necessary to store the old information related to the neighbours in order to make it available for updating the individual automaton itself. One of the approaches to this problem is to maintain two copies of the some data arrays, and to interchange their data after every step in the lattice gas cellular automaton evaluation.

In that algorithm the propagation phase implements according to a set of input data. Output information is first stored in data arrays  $nm$ ,  $nvx$  and  $nvy$  (the letter “n” denotes the “new” value, i.e. new value of the variable  $m$ ,  $vx$  and  $vy$ ), and then it moves back to the proper place in the arrays  $m$ ,  $vx$  and  $vy$ . Thus, this part of the algorithm ensures data transfer between pairs of arrays  $nm$  and  $m$ ,  $nvx$  and  $vx$ ,  $nvy$  and  $vy$ .

If the simulation model has graphical outputs the drawing of the initial system configuration takes place. Base on a value  $m$  in the lattice node, different colours assigned to different lattice nodes:

```

for (x=1; x<xmax; x++)
{
    for (y=1; y<ymax; y++)
    {
        m[x][y]=nm[x][y]; vx[x][y]=nvx[x][y]; vy[x][y]=nvy[x][y];
        putpixel (x, y, m[x][y]*print);
    }
}

```

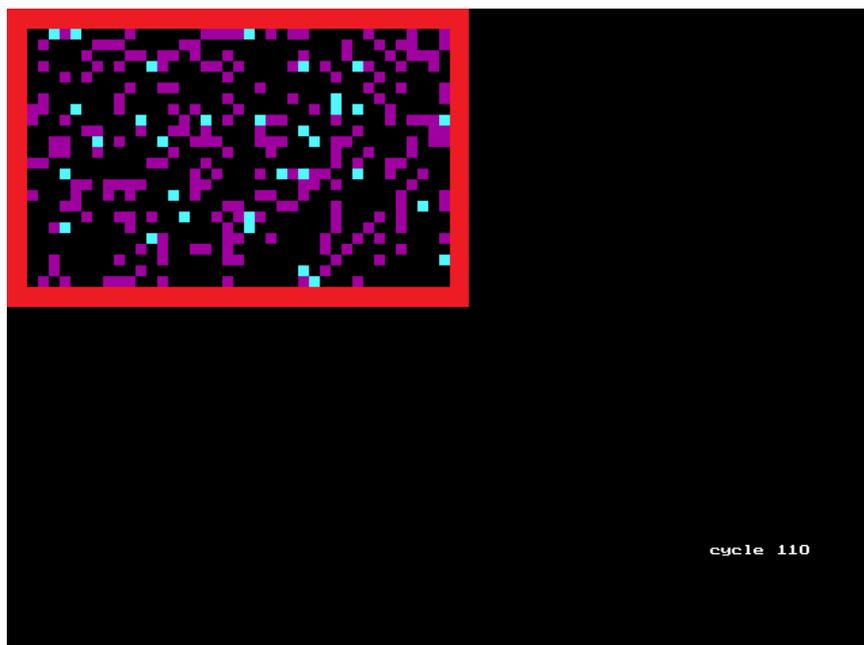
### 3.8. Code fragment 8 – Data arrays resetting

Before one cycle of the algorithm closes up, the resetting of some data arrays is needed, because new cycle will start and new set of data will be obtained. While passing through the lattice, algorithm checks whether the concrete lattice node is occupied by moveless particle or not. In a case that the lattice node is occupied by moving particle or it is empty the “resetting” of the information fields, which were used during the collision and propagation phases implementation, takes place:  $nm$ ,  $nvx$ ,  $nvx$ ,  $i1$ ,  $i2$ ,  $i3$ ,  $i4$ ,  $i5$ ,  $i6$  – all elements of them get value of 0. In other case the information, that the lattice node occupied by moveless particle, remains without a change in the data array  $nm$ :

```
for (x=3; x<xmax-2; x++)
{
  for (y=3; y<ymax-2; y++)
  {
    if (nm[x][y]<7)
      {nm[x][y]=0; nvx[x][y]=0; nvx[x][y]=0;
       i1[x][y]=0; i2[x][y]=0; i3[x][y]=0;
       i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;}
    else {nm[x][y]=7;}
  }
}
```

### 3.9. Code fragment 9 – Printout macro

Because the repeating of the cyclic part of the algorithm is equivalent to the one time step in the theory of cellular automata, the printout of carried out number of cycles is being used. The information about that is visible in the right bottom part of a monitor (see *Figure 31*).



**Figure 31:** Monitoring of the simulated system. The state after application of 110 cycles

### 3.10. Code fragment 10 – Final operations

The counting of cycles comes as a last operation of the cyclic part of the algorithm.

Before the main part of the algorithm will be closed, the last computer programming statements have to be called in order to finish the run of the algorithm. In C++ programming language they are as follows:

- *closegraph()* – ends the action of graphical functions in the program;
- *return()* – implements the ending of all functions that were called in a program, it also ends subroutines;
- *getch()* – by means of keyboard makes possible to finish the program run and return to the algorithm.

The time evolution of the FHP-1 LGCA model is evident from the *Appendix D*. A high value of the parameter *pco* ( $pco = 200$ ) is deliberately chosen. A movement of individual particles can thus be recorded. When a colour of particles changes from violet to blue, the two particle collision occurs. The evolution of the particle system was monitored for 20 time steps.

The basic skeleton of the Lattice Gas algorithm for a general-purpose computer that has been used for further introduced simulation experiments was described. Each particular simulation experiment includes for instance subroutines for extra conditions. These subroutines provide for example with certain particle monitoring or creation of the pressure gradient etc., ensure also a formation of special output data files. The concrete differences from the basic Lattice Gas Cellular Automata algorithm are described in particular simulation experiments (see Chapters 4-6).

## 4. VARIFICATION OF FHP-1 LGCA ALGORITHM FOR BROWNIAN MOTION

According to *Roache* [75], before any computer code is used to solve a complex problem, it must be verified in order to insure it has been implemented correctly. For that reason, a problem having an exact solution that encompasses most of the important physical phenomena must be chosen.

By using the developed algorithm discussed in the Chapter 3, some results obtained via verification tests are presented below (see Chapters 4.3 and 5.3). First, the Brownian motion simulation is used as a benchmark test for a verification of a newly developed FHP-1 LGCA algorithm.

### 4.1. Theoretical assumption

The *Brownian motion* was first discovered by Scottish botanist *Robert Brown* in 1827. He noticed a movement of plant pollens in water using microscope. But he was not able to determine the mechanisms that caused this motion. Later this was proved to be one of the effects of molecular motion and interactions between molecules [76].

Nowadays, *Brownian motion* is defined as a phenomenon whereby small particles suspended in a fluid tend to move in pseudo-random or stochastic paths through the fluid (liquid or gas), even if the fluid is calm and the drift vector is zero. This motion is caused by collisions between suspended particles and atoms or molecules of the fluid. The term “*Brownian motion*” also refers to a theory or model that is used to explain stochastic motion patterns. *Random walk*, in which the displacement of a particle is entire randomized, is an example of such a mathematical model [24]. *Random walk* has the Markov property, which means that the future state of the particle is determined by its current state only, not by any of past states (i.e. position of a moving particle at time  $t + 1$  depends only on its position at time  $t$ , and not on a path it took to get there).

According to *Feynman* [24], the logic question of the *Brownian motion* is: “Consider a little Brownian movement particle which is oscillates about because it is bombarded from all directions by randomly moving water molecules. After a given period of time, how far away is it likely to be from its original position?” Solution to this problem *Feynman* attributes to *Einstein* and *Smoluchowski* [24].

Let  $\mathbf{R}_n$  is the vector distance from the original position of the particle after  $n$  steps, then:

$$\mathbf{R}_n = \mathbf{R}_{n-1} + \mathbf{l} \quad (31)$$

where  $\mathbf{l}$  is a vector distance between two consecutive steps of the particle. The square distance is:

$$\mathbf{R}_n \cdot \mathbf{R}_n = R_n^2 = R_{n-1}^2 + 2\mathbf{R}_{n-1} \cdot \mathbf{l} + l^2 \quad (32)$$

After averaging over many trials,  $\langle R_n^2 \rangle = \langle R_{n-1}^2 \rangle + l^2$ , since  $\mathbf{R}_{n-1}$  and  $\mathbf{l}$  are not correlated and hence  $\mathbf{R}_{n-1} \cdot \mathbf{l} = 0$ . Thus the mean square value of the distance vector is proportional to the number  $n$  of steps:

$$R_n^2 = nl^2 = \frac{t}{\delta t} l^2,^{24} \quad (33)$$

where  $t$  is the time elapsed since the start of the *Brownian particle* motion and  $\delta t$  is the time elapsed between two successive steps. Since the number of steps is proportional to the time, the mean square distance is proportional to the time as well:

$$R_n^2 = \alpha t \quad (34)$$

The coefficient in the Equation (34) is usually expressed as  $\alpha = 2D$ . Coefficient “2” corresponds to the dimension and it is 6 for the 3D systems. The quantity  $D = \mu k_B T$  is the diffusion coefficient,  $\mu$  denotes the mobility coefficient (characterises the drift of molecules due to outside forces),  $k_B$  is the Boltzmann's constant and  $T$  is a absolute temperature. Then the mean square displacement of a *Brownian particle* in terms of the time elapses and the value of diffusivity becomes:

$$R_n^2 = 2Dt \quad (35)$$

## 4.2. FHP-1 Lattice Gas Cellular Automata algorithm for Brownian motion simulation

The basic FHP-1 LGCA algorithm, described in detail in the Chapter 3, is used here for a *Brownian motion* simulation to validate the algorithm. The difference between basic and modified algorithm is discussed in this chapter. An attention is paid to new parts of the algorithm code fragments or the most important differences. Code fragments, which are similar to the basic FHP-1 LGCA algorithm (see Chapter 3) are omitted from the description. The full code of the algorithm is presented in *Appendix E*.

### 4.2.1. Code fragment 1 – Header files and initialization of the simulation box

Compared to the basic FHP-1 LGCA algorithm presented in Chapter 3, few more arrays, variables and parameters are declared in this part of the algorithm for *Brownian motion* simulation:

- *code1, code2, code3, code4, code5, code6* – data arrays, where an exact position of the Brownian particle (a certain channel of the particular node) in every time step is being stored;

<sup>24</sup> Following the LGCA model,  $n$  is a number of time steps (units: *t.u.*), the distance between two neighbour nodes of the lattice(it is  $l$  in the equation) is equal to 1. The mean square distance of Brownian particle from its original position is linearly dependent on a time period of the simulation.

- $i$  – parameter that defines the appropriate place of the simulation box, where the *Brownian particle* is being generated;
- *fluid*, *boundary*, *hole*, *brownian* – parameters that determine a colour of a moving particles, a moveless particles and the *Brownian particle* at a graphical output. The empty node is named as a “hole” and at the graphical output it is black;
- *brownx* and *browny* – parameters that determine the initial position of the *Brownian particle* according to the 2D Cartesian coordinate system (XOY);
- $x1, x2, y1, y2$  –  $x$  and  $y$  coordinates of the *Brownian particle*. Index “1” indicates the position of the *Brownian particle* before collision and propagation phases. Index “2” belongs to the position after those phases implementation;
- *distance* – associated to the distance of the *Brownian particle* from its original position.

Compare to basic LGCA algorithm three more subroutines are declared in the algorithm. The collision and propagation phases of the *Brownian particle* supplying in subprograms are as follows:

```
int collisionbrown(void);  
float propagationoddbrown(void);  
float propagationevenbrown(void);
```

In addition to a graphical output, the data outputs are also stored. Therefore the data file is declared in the algorithm: `FILE *output0.`

### 4.2.2. Code fragment 4 – Occupation of channels by fluid particles

The *Brownian particle* is generated in this part of the algorithm in addition to all moving particles. The position of the *Brownian particle* is generated randomly and it is controlled by means of the parameter  $i$ . The parameter determines an acceptable distance from the centre of the simulation domain. The aim of this operation is to generate the *Brownian particle* randomly in the centre of the simulation domain.

In order to distinguish the *Brownian particle* from other fluid particles, its weight was increased by 13 mass units. Therefore the total mass in the node, where the *Brownian particle* occurs, is between 14 and 19 mass units (14 *m.u.* – the mass of the *Brownian particle*, 15...19 *m.u.* – the total mass in the node, where one *Brownian* and 1 to 5 fluid particles occur). It must be noted, that the mass of the *Brownian particle* is equal to one mass unit. In fact, its weight is the same as a weight of any moving particle. Its increasing by value of 13 is just the technical trick. It was used with the aim to distinguish the *Brownian particle* among other moving particles.

### 4.2.3. Code fragment 5-A – Data outputs

Before the cycling part of the algorithm starts, an initialisation of data files proceeds. File's name and its location are first given. The data are arranged into a  $m$ -by- $n$  matrix, where  $m$  is the number of rows (mostly the number of repeating cycles of the algorithm, i.e. the time of

the system evolution). Symbol  $n$  is the number of columns (the number of observed variables). Values of *cycle*, *brownx* and *browny*,  $x_2$ ,  $y_2$  and *distance* were saved into the output data file "*brown.cpp*".

#### 4.2.4. Code fragment 6 – The main cycle of the algorithm

The main cycle of the algorithm consists from the same steps as it was described in Chapter 3.6. Unlike the basic LGCA algorithm, there are two subroutines for collision phase implementation as well as for propagation phase. When the *Brownian particle* is identified in the lattice node, its collision phase is given by the subroutine *collisionbrown()* and propagation phase – by means of the subroutine *propagationbrown()*. Detailed description of those subroutines is presented in following Chapters 4.2.4.1 and 4.2.4.2.

##### 4.2.4.1. Code fragment 6-A – Collision phase

From the previous explanation it is obvious that the subroutine *collision()* serves for the implementation of collision phase between moving particles and is described in detail in the Chapter 3.6.1. Collisions between fluid moving and/or *Brownian particle* are implemented according to the *collisionbrown()* subroutine. It applies to the lattice node, where the total mass is  $> 13 m. u.$ , i.e. for the node, where the *Brownian particle* is detected.

The subroutine consists of nine analogous steps (see description of the code fragment 6-A in the basic FHP-1 LGCA algorithm – i.e. Chapter 3.6.1). In contrast to the fluid moving particles, the *Brownian particle* is the special one, because an exact position of it is being detected in every time step. For that reason the variable *brownp* is declared at the beginning of the subroutine.

The random marking of the *Brownian particle* takes place when the information about newly created state of the individual automaton (i.e. the occupation of individual channels  $i_1, i_2, i_3, i_4, i_5, i_6$  in the lattice node) is obtained. Marking of the *Brownian particle* consists of following steps:

1. Generation of a random number between 0 to 6, where the upper value is given by the number of channels in the lattice node:

$$\text{brownp} = \text{random}(6) .$$

2. Choosing the channel at random regarding to the value generated in a step 1. When the value  $n$  is generated, the channel  $n + 1$  is chosen. If the chosen channel after the collision's phase implementation gets the value of 1, i.e. becomes occupied, than the selected channel of the node is occupied by the *Brownian particle*. So, the value of the variable is increasing by 13. This information is being noted into the corresponding data array  $\text{code}_i$ <sup>25</sup> also:

---

<sup>25</sup> $\text{code}_i$  - it is  $\text{code}_1$ , or  $\text{code}_2$ , or  $\text{code}_3$ , or  $\text{code}_4$ , or  $\text{code}_5$ , or  $\text{code}_6$  in the algorithm

```
if ((brownp==0) && (i1[x][y]==1)) {i1[x][y]=13; code1[x][y]=13;}
```

In a case when the randomly generated channel is empty, go back to the step 1.

3. Repeating steps 1 and 2 as long as the newly position of the *Brownian particle* is being known.

It was mentioned before, in the Chapter 3.4.1, collision phase redistributes particles in an particular lattice node according to the collision rules only. The state of the LGCA is completely specified by indicating the occupied channels and empty ones. This implies that moving particles are indistinguishable and the Brownian one may randomly appears in the one of occupied channels in the particular node. The random motion of the *Brownian particle* is obtained then.

#### 4.2.4.2. Code fragment 6-B – Propagation phase

It was mentioned before, in the Chapter 3.6.2, the propagation phase is implemented separately in odd and even rows of the lattice. Propagation of fluid moving particles occurs in subroutines *propagationodd()* and *propagationeven()*, while the movement of the *Brownian particle* is realized in subroutines *propagationoddbrown()* and *propagationevenbrown()*. The principle of the last two subroutines is following:

1. Selecting the lattice node with coordinates  $x$  and  $y$ , where the *Brownian particle* is located. The individual channel occupation is of the particular interest now.
2. Coming subsequently through channels, denoted in the algorithm as  $i$ , of the lattice node and looking for the occupied ones.
3. If the channel  $i$  is occupied by moving particle ( $i1[x][y]=1$ ) or a Brownian one ( $i1[x][y]=14$ ), detecting the state of the neighbour node, which communicates with the selected lattice node through the channel  $i$ .
4. Relocation of the particle setting trough the channel  $i$  in direction towards the neighbouring node in the case when the neighbour node is not occupied by a moveless particle. The new particle's number (the value of the parameter  $nm$ ) in the neighbouring node extends by the value 1 and by the value of the variable  $code_i$ . If the *Brownian particle* relocates,  $code_i$  gets value 13, in otherwise the value remains 0. New values of  $x$  component  $nvx$  and  $y$  component  $nvx$  of the total particles velocity in the neighbour node are extended by the value of the  $x$  and  $y$  component of the velocity of particle coming through the channel  $i$ .
5. Implementation of the bounce-back type of reflective boundary condition when the neighbouring node of the selected lattice node is occupied by solid particle. Thus, the new number of particles (the value of the parameter  $nm$ ) in the chosen lattice node extends by value 1 and by the value of the variable  $code_i$ . New values of  $x$  component  $nvx$  and  $y$  component  $nvx$  of the total particles velocity in the lattice node are extended by the values of  $x$  and  $y$  component of the velocity of particle

coming inside the selected lattice node through the channel  $i$ . In other words instead of the displacement of the particle from the channel  $i$  to the neighbouring node, it moves back to the selected lattice node and all the information connected with this particle is being hold within.

The implementation of steps 3-5, when the channel  $i1$  is denoted as an occupied is shown here:

```

if ((i1[x][y]==14) || (i1[x][y]==1))
{
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1+code1[x][y];
        nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle; x2=x; y2=y;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1+code1[x][y]; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
        nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle; x2=x-1; y2=y-1;}
}

```

6. Repeating steps 4 and 5 for all occupied individual channels of the selected lattice node.
7. Detecting the new position of the *Brownian particle*. Coordinates  $x$  and  $y$  of the node, where the *Brownian particle* was shifted, are recorded.

#### 4.2.5. Code fragment 9 – Printout macro

Data outputs are presented as data files. The current distance of *Brownian particle* from its initial position is calculated according to the Pythagorean theorem:

$$\text{distance}=\text{sqrt}(\text{pow}(x_2-\text{brown}x,2)+\text{pow}(y_2-\text{brown}y,2)),$$

where  $\text{brown}x$  and  $\text{brown}y$  are  $x$  and  $y$  coordinates of the *Brownian particle*'s at the initial position,  $x_2$  and  $y_2$  – coordinates of the current position.

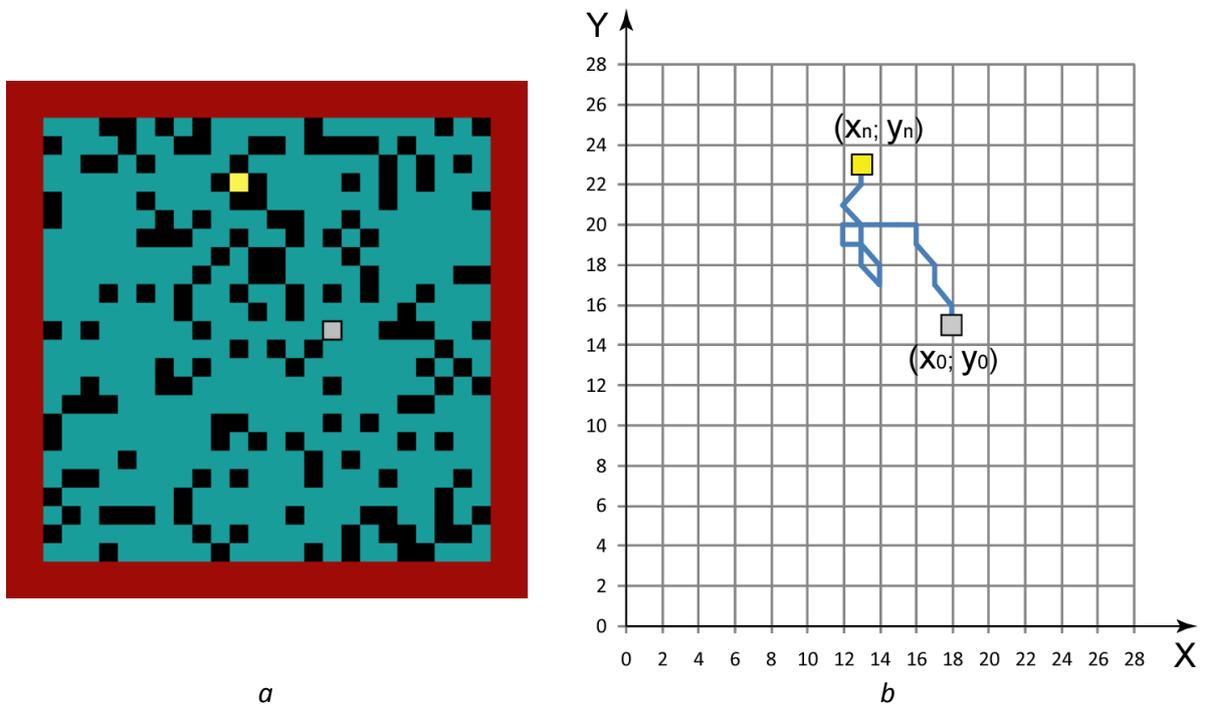
If the *Brownian particle* collides with the solid boundaries of the simulation domain, the simulation is imediatly finished and data outputs are not included into the final data treatment. Such a simulation is deliberately broken and thus can not be treated together with other data due to missing output files. Output files are saved only when the simulation is properly finished.

### 4.3. Simulation setup

The modified FHP-1 LGCA algorithm was applied for a two-dimensional *Brownian motion* simulation. The reduced simulation domain of a size  $N_x \times N_y = 30 \text{ l.u.} \times 30 \text{ l.u.}$ , where  $N_x$  and  $N_y$  are numbers of nodes in  $x$  and  $y$  directions of the lattice is shown in the *Figure 32*. From the *Figure 32 (a)* it is evident that simulation domain is bordered by solid walls. Inside the simulation domain moving particles are generated with a certain probability. Subsequently, the lattice gas of average density  $\rho = 3$  or  $1,5$  particles per one lattice node is applied. In the *Figure 32* when the lattice node is empty it is black. Red colour is used for the

identification of moveless particles, blue one denotes moving particles. Initial position of the Brownian particle is presented by the grey square, and its final position after 20 time steps – by yellow one. In a real simulation in the middle of the simulation domain one of the fluid moving particles is marked as a Brownian one. Trajectory of the Brownian random motion during 4000  $t.u.$  is monitored. The bounce back type of fluid particle's reflection is applied to the solid boundaries of the simulation domain. The exact simulation setups are presented in the *Table 2*. The table header includes the following parameters:

- $pco$  – parameter, which is used in the algorithm, it denotes the probability of channel occupation by moving particles;
- $x_0$  and  $y_0$  – it is  $x$  and  $y$  coordinates of the initial position of the Brownian particle;
- Time – it is the total time period of the simulation.



**Figure 32:** Simulation of the Brownian motion presented on the reduced simulation domain 30 l. u.  $\times$  30 l. u.: **a** – the simulation domain bounded by solid walls (red lines), moving particles (blue squares), initial position of the Brownian particle (grey square) and its final position (yellow square) after 20 time steps, black squares present empty lattice nodes; **b** – Brownian random motion during 20 time steps obtained by the developed model based on the FHP-1 LGCA model

From the *Table 2* it is evident that simulation settings are different due the lattice gas density and initial position of the *Brownian particle*. The initial position of the *Brownian particle* is generated randomly in every simulation. Thus the results obtained from the simulation should be independent on the initial position of the *Brownian particle*.

As it was explained in the Chapter 4.2.5, the distance of the *Brownian particle*  $R$  (see *Figure 33*) from its initial position after  $n$  time steps is calculated as:

$$R_n = \sqrt{(x_n - x_0)^2 + (y_n - y_0)^2} \quad (36)$$

Where  $x_n$  and  $y_n$  are  $x$  and  $y$  coordinates of the Brownian particle's current position;  $x_0$  and  $y_0$  are coordinates of its initial position.

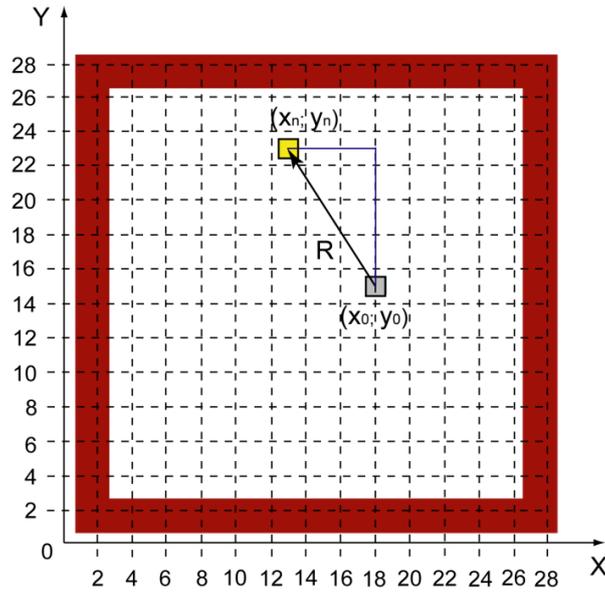


Figure 33: Displacement  $R$  of the Brownian particle

Results of two simulation experiments are reviewed below. Every simulation experiment was repeated ten times for ten different initial positions of the *Brownian particle*, that was randomly generated. In order to obtain the numerical results, the displacement of the *Brownian particle* was averaged over 7 experiments. Three simulations in each experiment were broken off because the *Brownian particle* collided with solid boundaries of the simulation domain. The size of the simulation domain was limited by the monitor resolution.

Table 2: The list of Brownian motion computer simulations and their setups

	Size of the simulation domain, l. u. x l. u.	Average density, m.u./l.u.	$x_0$ , l.u.	$y_0$ , l.u.	Time, t. u.	Data output files
Simulation experiment 1	300 x 300	3	164	144	4000	BROWN03.CPP
	300 x 300	3	158	156	4000	BROWN04.CPP
	300 x 300	3	160	164	4000	BROWN05.CPP
	300 x 300	3	170	152	4000	BROWN06.CPP
	300 x 300	3	-	-	-	-
	300 x 300	3	164	128	4000	BROWN08.CPP
	300 x 300	3	-	-	-	-
	300 x 300	3	-	-	-	-
	300 x 300	3	142	157	4000	BROWN11.CPP
	300 x 300	3	166	157	4000	BROWN12.CPP
Simulation experiment	400 x 400	1,5	170	181	4000	BROWN01.CPP
	400 x 400	1,5	209	191	4000	BROWN02.CPP
	400 x 400	1,5	-	-	-	-
	400 x 400	1,5	227	196	4000	BROWN04.CPP

400 x 400	1,5	228	182	4000	BROWN05.CPP
400 x 400	1,5	-	-	-	-
400 x 400	1,5	211	166	4000	BROWN07.CPP
400 x 400	1,5	222	237	4000	BROWN08.CPP
400 x 400	1,5	219	183	4000	BROWN09.CPP
400 x 400	1,5	-	-	-	-

This table includes value of the following parameters:

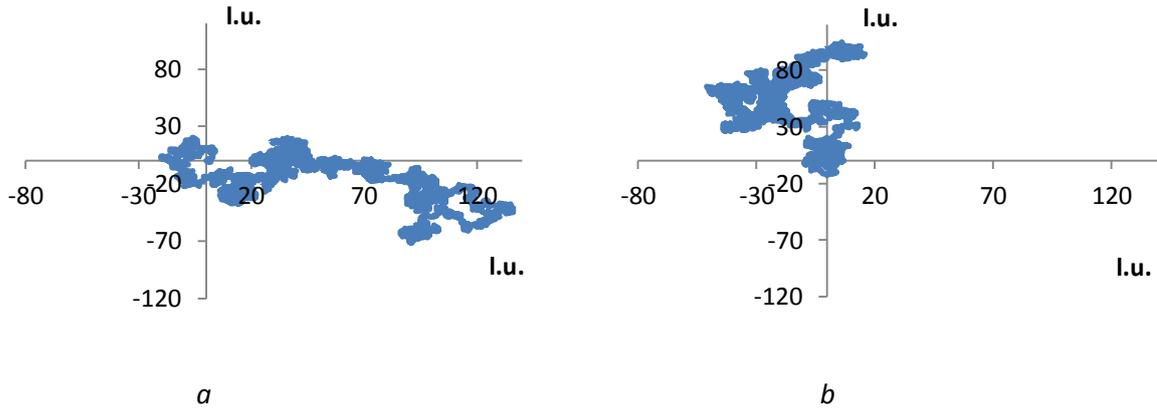
- Size of the simulation domain - it is presented by the length  $L$  x the width  $d$ ;
- Average density – corresponds to the average number of moving particles in the lattice node;
- $x_0$  and  $y_0$  - corresponds to the  $x$  and  $y$  coordinates of the initial position of the *Brownian particle*;
- Time of the simulation – it is the total time period of the simulation;
- Data output files – are data files obtained from the computer simulation.

#### 4.4. Results and discussion

The time evolution of the FHP-1 LGCA model for *Brownian motion* simulation inside the reduced simulation domain of a size  $N_x \times N_y = 30 \times 30$  l.u. is presented in an *Appendix E*. This scaled down version of the simulation was used for a graphical representation of the *Brownian particle* movement only. The state of the simulation system is detected here after every time step during  $t = 20$  t.u. The square lattice with coordinates  $x$  and  $y$  is depicted on pictures for simplified representation of the results. But according to the principles of the FHP-1 LGCA model computer simulation was performed at the hexagonal Bravais lattice.

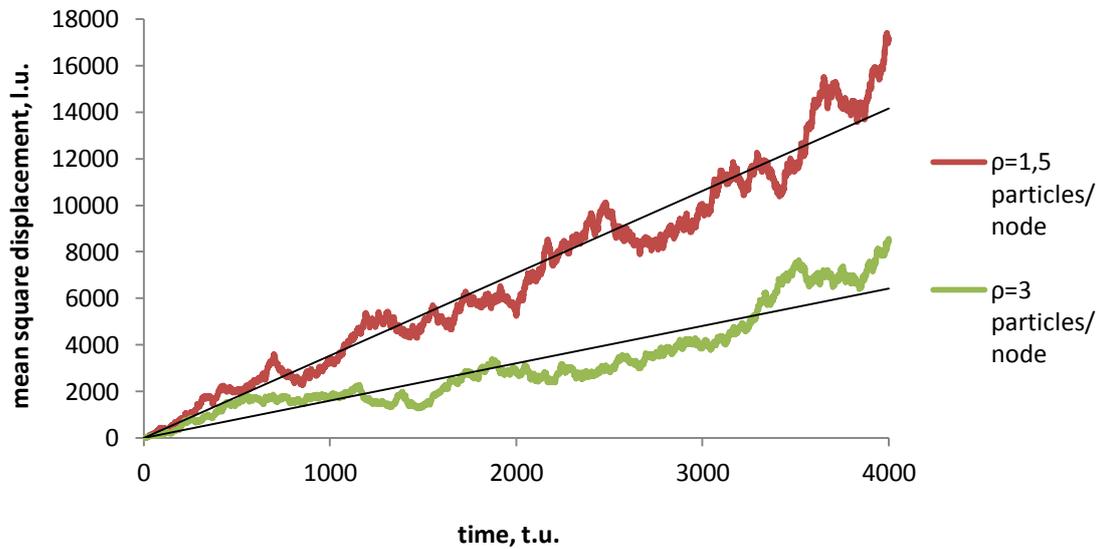
The exact paths of the *Brownian particle* over  $t = 4000$  t.u. were monitored and are presented in *Appendix G* and *Appendix H*. Those paths aren't linear. They are often similar to a "bonsai tree" shape, where some part of the path is approximately linear and another part is an area, where the *Brownian particle* is rather going back or turning in a small closed area. The two most sequence shapes of the *Brownian particle's* paths are presented in the *Figure 34*.

*Figure 35* plots the mean square displacement  $R^2$  of the *Brownian particle* from its initial position as a function of time. As was mentioned before, the square displacement was averaged over 7 simulations. Nevertheless, fluctuations around the linear trend line are still evident. The greater degree of fluctuations was recorded in the simulation experiment with a lower lattice gas density ( $\rho = 1,5$  particles/node). The higher density of the lattice gas is simulated, the smaller straight forward displacements of the *Brownian particle* are achieved.



**Figure 34:** Paths of the Brownian particle after 4000 time steps: **a** – the straight type of paths (simulation experiment 1, data output BROWN04.cpp); **b** – the “bonsai tree” shape of the path (simulation experiment 1, data output is BROWN11.CPP)

A first benchmark test for a verification of a newly developed FHP-1 LGCA algorithm was being considered as a successful one. Better agreement is achieved using higher value of lattice gas density. For more accurate simulation of the Brownian motion the biggest size of the simulation domain, longer time of the Brownian particle monitoring or averaging over the larger number of simulations are recommended.



**Figure 35:** The main square displacement of the Brownian particle as a function of time, for  $\rho=1,5$  particles/node and 3 particles/node

## 5. VARIFICATION OF THE FHP-1 LGCA ALGORITHM FOR POISEUILLE FLOW

The 2D *Poiseuille flow* is the simplest kind of flow system that can be simulated using FHP-1 Lattice Gas Cellular Automata. It is an incompressible flow between two stationary parallel plates and driven by constant body force [24]. According to data from a number of dissertations [77, 78] it is evident that Poiseuille flow simulation is being used as a benchmark test for a verification of a newly developed algorithm, if it is intended to transport phenomena modelling using Lattice Gas Cellular Automata or Lattice Boltzmann approach. It is also used as verification for numerical analysis since the analytical solution can be obtained from *Navier-Stokes equation*. *Poiseuille flow* model requires only the bounce-back type of boundary reflections along the walls of a channel and periodic boundary conditions in the flow direction.

### 5.1. Theoretical assumption

Poiseuille flow is an example of an elementary fluid flow. It is also a simple model for flow through a crack or joint of a rock. Fluid flow through a porous media, and especially through fibrous materials, is a subject of wide interest in textile branch. The textile industry encounters with this phenomenon during a lot of production and finishing processes. It is also a subject of study from the textile comfort properties point of view. Permeability is the physical parameter of prime interest in these circumstances. Moreover, the permeability measurement is one of the most important ways that enable to evaluate final textile products for its application. For example, permeability is a critical parameter for the application of fibrous materials as filters, barrier materials, sportive clothing, etc. Invention of multilayer textile materials is based on an idea to combine various layers with different permeability to reach an optimal comfort with respect to the water vapour transport outward and retention of external liquid droplets [17].

Generally, fluid flow is a three-dimensional process, but it can be reduced in some cases to the two-dimensional due to its symmetry. There is a number of authors who studied Poiseuille fluid flow under various conditions using Lattice Gas Cellular Automata or Lattice Boltzmann models. For example, *Rothman* in his work [79] studied two-dimensional Poiseuille flow as a function of the variety of channel thickness and variety of pressure gradient. The similar computer simulation experiment was done by *Wolf-Gladrow* [31]. The same dependence was of *Chen's* interest [80] for three-dimensional channel flow. Interesting problems were solved by *Yang* few years ago [81]. It was based on the Lattice-Boltzmann model, where the influence of various interactions between the fluid and channel walls was considered. Particularly, one part of the channel surface was wetted by a liquid while other parts repelled it. Using two Poiseuille flows, opposite to each other, *Kadanoff et al.* developed the method to build a numerical viscometer [82].

All above mentioned works, dealing with computer simulations, first prove the parabolic velocity profile of the flow. The velocity  $\mathbf{v}$  of such fluid flow is everywhere parallel to the channel walls if the uniform pressure gradient is applied along the two-dimensional channel. Thus, the  $y$  component of the flow velocity is zero. The  $x$  component of velocity is strongly influenced by the interaction of the fluid with walls of the channel. The velocity  $\mathbf{v}$  of a viscous fluid is considered to be zero at solid boundaries, when no-slip boundary conditions are used. The maximum velocity value appears in the centre of the channel, because of viscous forces inside the fluid.

According to *Rothman* [79] this type of the flow is being known as a *plane Poiseuille flow* and it is being governed by equation:

$$v_x(y) = \frac{G}{2\mu} \left( \frac{d^2}{4} - y^2 \right) \quad (37)$$

where  $G = -\frac{dp}{dx}$  is a pressure gradient,  $\mu$  is a dynamic viscosity value of the fluid,  $d$  is a distance between two parallel plates (in other words it is the channel width).

To find the volumetric flow rate of flow per unit area  $q$  it is necessary to integrate  $v_x$  from  $y = -\frac{d}{2}$  to  $y = \frac{d}{2}$  and divide by the unit area – i.e. by the channel width  $d$ , then:

$$q = \frac{Gd^2}{12\mu} \quad (38)$$

Equation (38) is in accordance with *Darcy's law* known from the middle of the 19<sup>th</sup> century, when French Henry Darcy experimentally discovered that the flow rate through a porous medium, including a fibrous one, is linearly proportional to the applied pressure gradient. For a flow along the  $x$  axis of the channel it holds:

$$q = -\frac{k}{\mu} \frac{dp}{dx} \quad (39)$$

where  $k$  is the permeability of the medium. From Equations (38) and (39) it is evident, that the permeability of the channel with two parallel plates at the distance  $d$  is  $k = \frac{d^2}{12}$ .

*Darcy's law* is valid for laminar flows, where the Reynolds number is relatively small. In other words, the law is valid for *steady Poiseuille flows* with parabolic velocity profiles in free channels.

## 5.2. FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow simulation

The algorithm based on the FHP-1 LGCA model which developing was described in detail in the Chapter 3, is used for a *Poiseuille flow* simulation. In contrast to the developed basic algorithm, this algorithm allows the computer simulation of the fluid flow inside the infinite channel in the  $x$  direction. The infinity is given by periodic boundary conditions. Reflective boundary conditions are used at the top and down channel boundaries. Pressure gradient is

applied in order to create the flow inside the channel. As a result, computer simulation provides information about flow velocity inside the channel.

Thus, for the second verification test few new parts of the algorithm are developed. Those parts are described in details in this chapter. Code fragments, which are similar to the basic FHP-1 LGCA algorithm (see Chapter 3) are omitted from the description. The full code of the algorithm is presented in *Appendix I*.

### **5.2.1. Code fragment 1 – Header files and initialization of the simulation box**

Compared to the basic FHP-1 LGCA algorithm, few special variables and parameters were declared in this part of the algorithm for *Poiseuille flow* simulation:

- *mass* and *velocity* – are being used in calculation of fluid moving particles and a flow rate;
- *node* – counts the number of lattice nodes inside the simulation domain, where the fluid particles are moving;
- *force* – parameter that defines the probability of the force creation along one boundary of the lattice;
- *ventilator* – parameter that determines the size of an imaginary ventilator, i.e. the area (number of boundary columns), where the force was created;
- *transfer14*, *transfer25*, *transfer36* – record the change of the *x* component of fluid particle momentum in a position of the imaginary ventilator after the forced shifts of moving particles from channels *i1*, *i2*, *i3* to channels *i4*, *i5*, *i6*;
- *V*– is a data array, where the velocity streamlines are recorded.

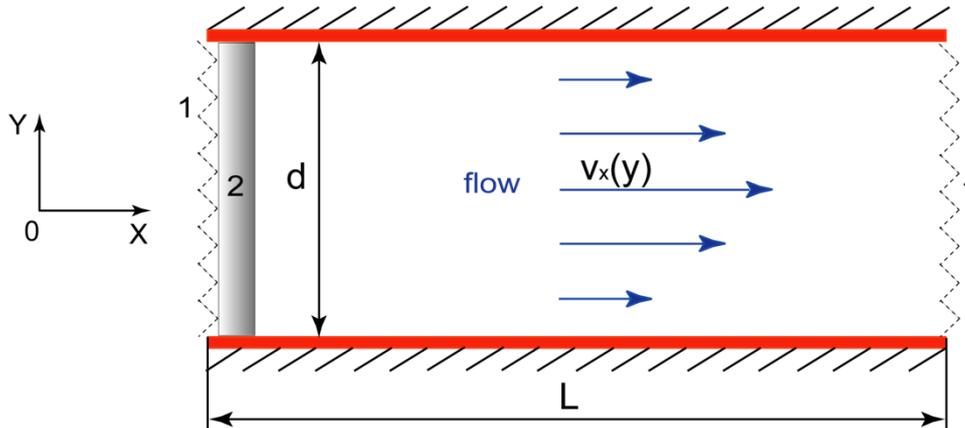
Special subroutines are declared in the algorithm. The propagation phase is realized in four subprograms and is applied at boundaries of the channel in accordance to odd and even rows of the lattice:

- *propagationleftsideodd()* – propagation of the fluid moving particles in all odd rows at the left boundary of the channel;
- *propagationleftsideeven()* – propagation of the fluid moving particles in all even rows at the left boundary of the channel;
- *propagationrightsideodd()* – propagation of the fluid moving particles in all odd rows at the right boundary of the channel;
- *propagationrightsideeven()* – propagation of the fluid moving particles in all even rows at the right boundary of the channel;.

Activity of the imaginary ventilator is created in the subprogram *turnright()*. Velocity profile of the fluid is calculated in a subprogram *profile()*. Two output data files are declared in the algorithm: `FILE *output0` and `FILE *output1`.

### **5.2.2. Code fragment 3 – Creation of the simulation domain**

The channel with upper and bottom solid boundaries is created by means of generation the moveless type of particles. At the right and left sides of the simulation domain no moveless particles are generated, periodic boundary conditions are applied here (see *Figure 36*). The length of the channel is  $L = 550 \text{ l.u.}$ , the width  $d$  ranging from  $25\sqrt{3}/2$  to  $100\sqrt{3}/2$  with increments  $25\sqrt{3}/2$  for particular simulations.



**Figure 36:** The geometry of two-dimensional channel for Poiseuille flow simulation: **1** – periodic boundary conditions, **2** – the imaginary ventilator,  $L$  is the length and  $d$  is the width of the channel

### 5.2.3. Code fragment 5-A – Data outputs

Before the cycling part of the algorithm the data file initialisation starts. File's name and its location is given first. Values of variables *transfer14*, *transfer25*, *transfer36*, *cycle*, *node*, *mass* and *flow* are saved into the data file "*FLOW.CPP*".

### 5.2.4. Code fragment 6 – The main cycle of the algorithm

The structure of the main cycle of the algorithm is evident from the flowchart presented in *Figure 37*. It is obvious, that during every time step algorithm goes through all nodes of the lattice. When the moving type of particles is located in the node, collision and propagation phases take place. The force sifts of the moving particles in the direction of flow occurs, when they are located in a position of the imaginary ventilator. Operations needed for output data obtaining are not illustrated at the flowchart. Calculation of the flow rate occurs in every time step. The velocity profile is being calculated after the steady state of the flow is obtained.

Detailed description of the collision phases was presented in the Chapter 3.6.1. This algorithm uses the the same subprogram *collision* (). The propagation phase occurs here inside the simulation domain as well as its boundaries. This fact is reflected into the subprograms which are determined for the propagation phase implementation. It is explained below in the Chapter 5.2.4.2. The principlne of the force shifts of moving particles at left boundary of the simulation domain is described in Chapter 5.2.4.1.

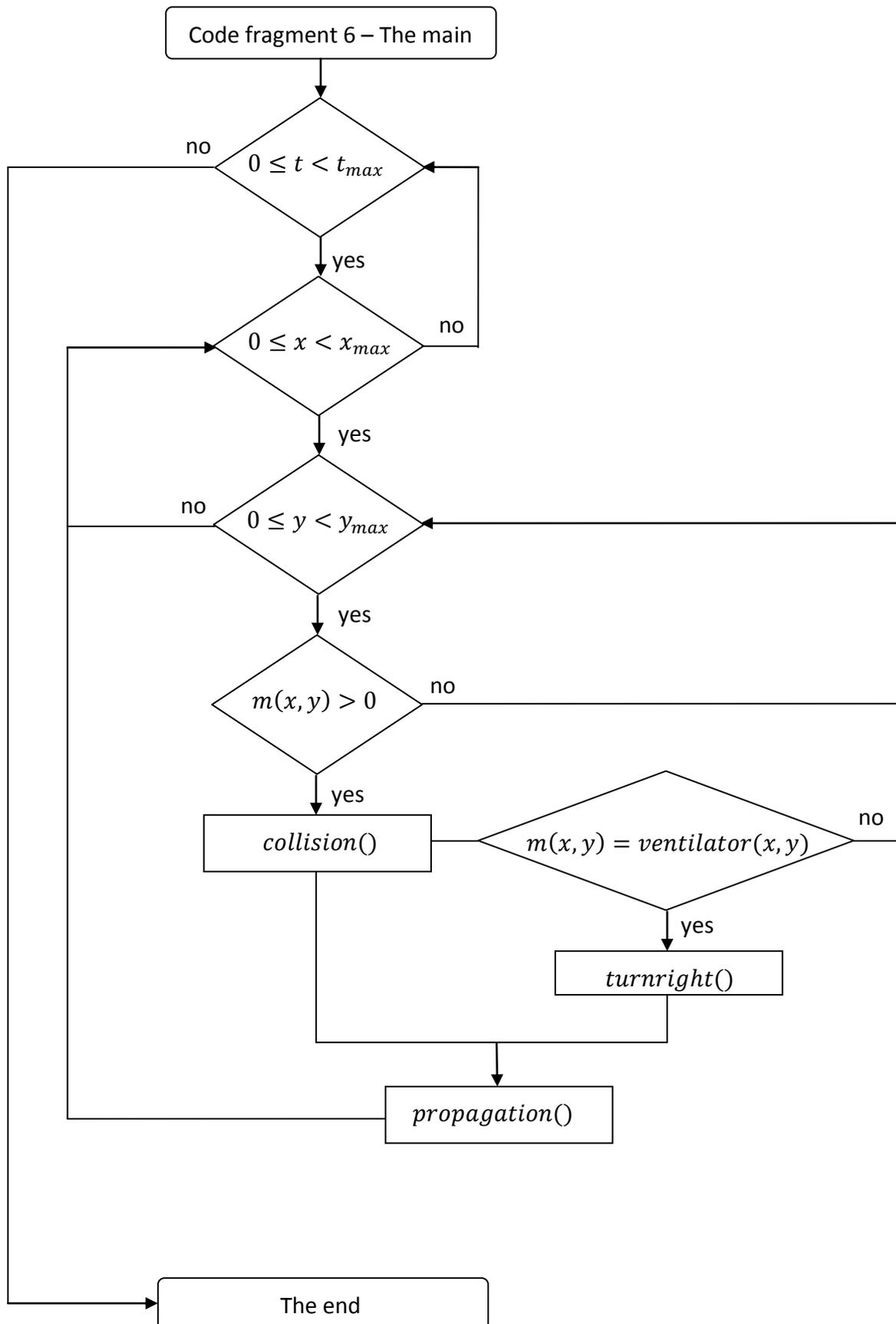
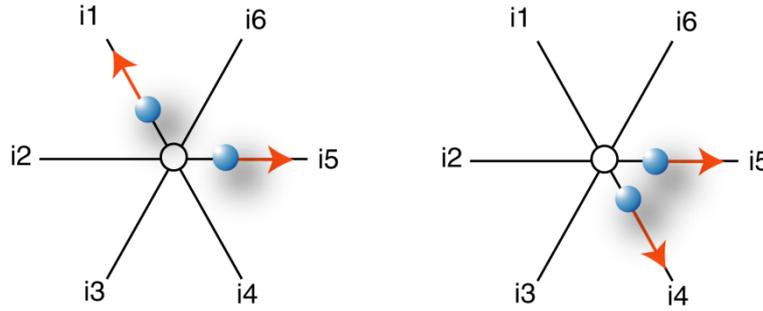


Figure 37: The flowchart representing the main cycle of the algorithm developed for a simulation of the Poiseuille flow

5.2.4.1. Code fragment 6-B – Pressure gradient

Pressure gradient is created in the algorithm by means of a force equally applied along left boundary of the channel in a position of the imaginary ventilator with a certain probability. The process of the force creation given by the subroutine *turnright()*. This operation relates to every lattice node at the left boundary of the channel and consists of following steps:

1. Going through channels in a particular lattice node and successively choosing the pair of opposite channels.
2. First choosing the pair of channels *i1* and *i4*.
3. Propagation the fluid moving particle from the channel *i1* to the channel *i4* if the channel *i1* is occupied by fluid moving particle and the channel *i4* is empty (see *Figure 38*). As a result of this operation the value *transfer14* is reduced by value 1 (the reason is explained in this chapter below).
4. Repeating steps 2 and 4 for pairs of channels *i2* and *i5*, *i3* and *i6*.



**Figure 38:** An example of the forced reorganization of channel occupation. Propagation of moving particle form the channel *i1* to *i4*

As it was mentioned in the Chapter 5.2.1, *transfer14*, *transfer25*, *transfer36* correspond to the change in the *x* component of fluid momentum according to the reorganization of channel occupation. If  $\Delta p_{x14}$  is a change in the *x* component of momentum at a single lattice node as a result of fluid particle's shifting from the channel *i1* to the channel *i4*, then:

$$\Delta p_{x14} = m\Delta v_{x14} = m(v_x - v_{xi1} + v_{xi4}) = m(v_x - (-\cos 60) + \cos 60) = m(v_x + 1),$$

Where *m* is a mass (i.e. the number of particles, the mass of each particle is equal to 1) *v<sub>x</sub>* is a *x* component of total velocity in a lattice node and its consists of  $v_x = v_{xi1} + v_{xi2} + v_{xi3} + v_{xi4} + v_{xi5} + v_{xi6}$ ;  $v_{xi1} \dots v_{xi6}$  are *x* component of velocity in accordance of channels occupation.

Similarly, the change in the *x* component of momentum  $\Delta p_{x36}$  at a single lattice node after shifting the fluid moving particle from the channel *i3* to the channel *i6* is:

$$\begin{aligned} \Delta p_{x36} &= m\Delta v_{x36} = m(v_x - (-v_{xi3}) + v_{xi6}) = m(v_x - (-\cos 60) + \cos 60) \\ &= m(v_x + 1) \end{aligned}$$

The change in the  $x$  component of momentum  $\Delta p_{x25}$  at a single lattice node after shifting the fluid moving particle from the channel  $i2$  to the channel  $i5$  is:

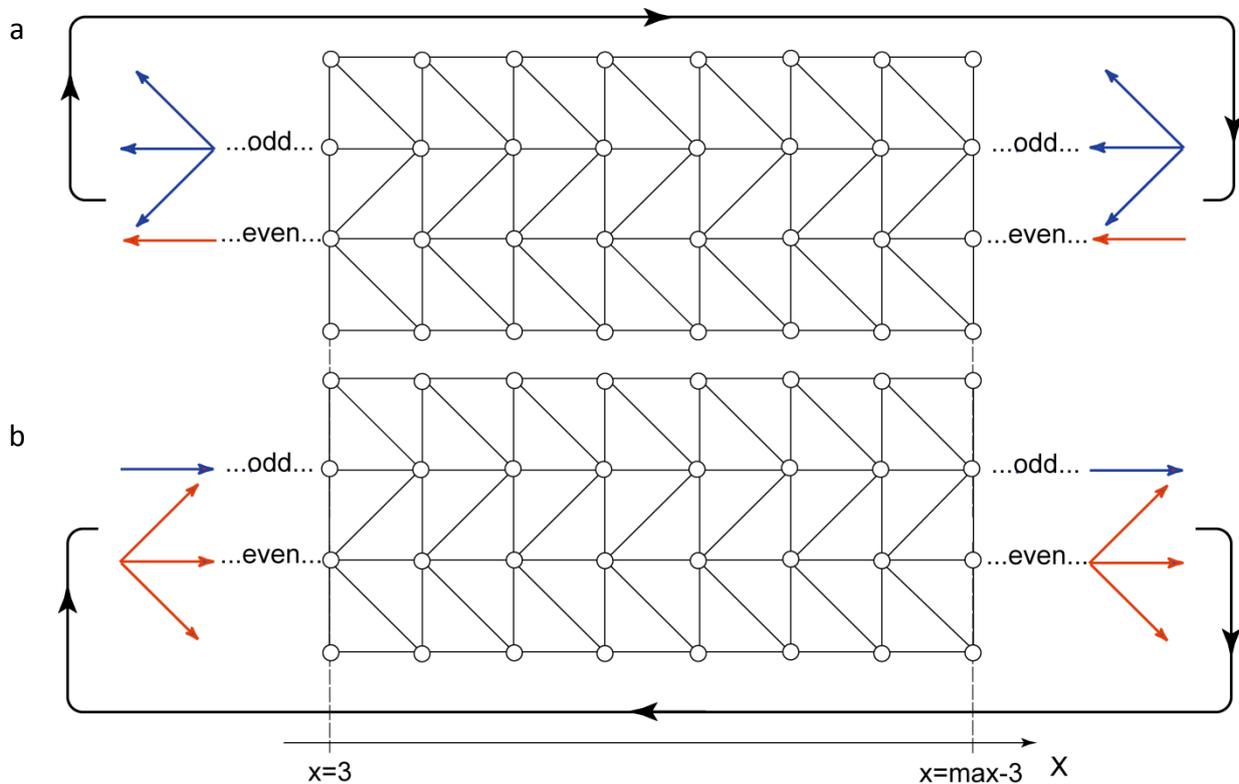
$$\Delta p_{x25} = m\Delta v_{x25} = m(v_x - (-v_{xi2}) + v_{xi5}) = m(v_x - (-1) + 1) = m(v_x + 2)$$

Thus, parameters  $transfer14$ ,  $transfer25$ ,  $transfer36$  reflect an increasing of the value  $v_x$  by increment equal to 1 or 2. In contrast to  $\Delta p_{x14}$  and  $\Delta p_{x36}$  the change in the  $x$  component of momentum  $\Delta p_{x25}$  is equal to 2.

**5.2.4.2. Code fragment 6-C – Propagation phase**

The propagation phase is implemented separately in odd and even rows of the lattice according to the basic FHP-1 LGCA algorithm. Furthermore, periodic boundary conditions require the special subprograms for propagation phase implementation. So, propagation of moving particles inside the channel occurs in subroutines *propagationodd()* and *propagationeven()*, while the propagation at left and right boundaries in the simulation domain – in subroutines *propagationleftsideodd()* and *propagationleftsideeven()*, *propagationrightsideodd()* and *propagationrightsideeven()*.

The principle of the propagation phase at boundaries of the channel does not differ from the propagation inside the channel bulk. The principle of the propagation phase was described in details in the Chapter 3.6.2. But the special conditions are defined at the left and right boundaries of the lattice (see *Figure 39*).



**Figure 39:** Propagation of moving particles at the left (a) and at the right (b) boundaries of the channel. Periodic boundary conditions are applied

Lattice nodes which are located at the left boundary of the channel (i.e.  $x = 3$ ), in all odd rows of the lattice communicate through channels  $i1$ ,  $i2$  and  $i3$  with lattice nodes at the right boundary (i.e.  $x = x_{max} - 3$ ). This communication realized through the channel  $i2$  in even rows of the lattice. The ordering of channels according to the lattice geometry was illustrated in *Figure 28* (see Chapter 3.4.1). Similarly, at the right boundary of the channel ( $x = x_{max} - 3$ ) lattice nodes communicates with their neighbours at the left boundary ( $x = 3$ ) through channels  $i4$ ,  $i5$  and  $i6$  in all even rows. It is channel  $i5$  in odd rows of the lattice.

### **5.2.5. Code fragment 9 – Printout macro**

This code fragment includes two parts. First, the  $x$  component of the flow rate of the lattice gas is calculated as a ratio between  $x$  component of the velocity and a number of all moving particles (i.e. their total mass). This computation takes place in every time step and the output data is being recorded into the data file *FLOW.CPP*. The knowledge of the flow rate is important to determination the steady state of the flow. An acquiring of fluid velocity profile starts after the steady state of the flow is achieved.

Velocity profile of the fluid represents the  $x$  component of the particle velocity averaged over the length of the channel for each  $y$  coordinate of the lattice and additionally averaged over time during the steady state of the flow. A subprogram *profile()* is used in order to determine the velocity profile of the fluid. The calculation consists of following steps:

1. Coming through the simulation domain (excepting rows of the lattice with moveless particles and lattice nodes where imaginary ventilator takes place) and calculating:
  - the sum of  $x$  component of particle's velocity in a lattice row;
  - the number of moving particles in a lattice row.
2. Calculation the average  $x$  component of particle's velocity.
3. Repeating steps 1 and 2 for every lattice row of the simulation domain.

The velocity of the fluid is also averaged in time in order to obtain more accurate results. After every time step its value is stored into the data array named as  $V$ . The final value of the velocity is calculated before final operations of the algorithm and is saved as an output data file *PROFILE.CPP*.

## **5.3. Simulation setup**

The two-dimensional channel geometry is employed in order to suit all computer simulations. Overhead and bottom channel sides are composed of solid walls (moveless particles) that restrict the flow in a perpendicular direction of the channel. The length of the channel  $L$  is chosen to be 550 lattice units (*l.u.*), but due to usage of the periodic boundary conditions the infinitely long channel in  $x$  direction is in fact created. Fluid particles are generated into the free space between solid walls. Lattice gas density  $\rho$  is chosen to be 2,5

particles per node and this condition is used in each simulation. The bounce-back type of reflective boundary conditions is used when collisions between moving and moveless particles took place. This type of reflection was applied at overhead and bottom channel sides. The behaviour of the flow as a function of scale is studied. The width of the channel  $d$  and the probability of force creation  $pf_c$  at the left boundary of the channel varied. Different pressure gradients  $G = -\frac{dp}{dx}$  is created along the channel according to the value of the  $pf_c$ . The exact simulation setups are presented in the *Table 3*.

*Table 3: The list of Poiseuille flow computer simulations and their setups*

	Size of the channel, l. u.		Average density, m.u./l.u.	force / $pf_c$	Parameter $f_x$	Time of the simulation, t. u.	Steady state of the flow, t. u.
	The length $L$	The width $d$					
1.	550	$25\sqrt{3}/2^{26}$	2,5	100 / 1	2	10000	5000
2.	550	$25\sqrt{3}/2$	2,5	200 / 0,5	1,4	10000	5000
3.	550	$25\sqrt{3}/2$	2,5	1000 / 0,1	0,4	10000	5000
4.	550	$25\sqrt{3}/2$	2,5	2000 / 0,05	0,2	10000	5000
5.	550	$25\sqrt{3}/2$	2,5	10000 / 0,01	0,03	10000	5000
6.	550	$50\sqrt{3}/2$	2,5	100 / 1	2	10000	5000
7.	550	$50\sqrt{3}/2$	2,5	200 / 0,5	1,4	10000	5000
8.	550	$50\sqrt{3}/2$	2,5	1000 / 0,1	0,4	10000	5000
9.	550	$50\sqrt{3}/2$	2,5	2000 / 0,05	0,2	10000	5000
10.	550	$50\sqrt{3}/2$	2,5	10000 / 0,01	0,03	10000	5000
11.	550	$75\sqrt{3}/2$	2,5	100 / 1	2	10000	5000
12.	550	$75\sqrt{3}/2$	2,5	200 / 0,5	1,4	10000	5000
13.	550	$75\sqrt{3}/2$	2,5	1000 / 0,1	0,4	10000	5000
14.	550	$75\sqrt{3}/2$	2,5	2000 / 0,05	0,2	10000	5000
15.	550	$75\sqrt{3}/2$	2,5	10000 / 0,01	0,03	10000	5000
16.	550	$100\sqrt{3}/2$	2,5	100 / 1	2	10000	5000
17.	550	$100\sqrt{3}/2$	2,5	200 / 0,5	1,4	10000	5000
18.	550	$100\sqrt{3}/2$	2,5	1000 / 0,1	0,4	10000	5000
19.	550	$100\sqrt{3}/2$	2,5	2000 / 0,05	0,2	10000	5000
20.	550	$100\sqrt{3}/2$	2,5	10000 / 0,01	0,03	10000	5000

The table includes values of following parameters:

- Size of the channel - it is presented by its length  $L$  and width  $d$ ;
- Average density – corresponds to the average number of moving particles in the lattice node;
- *force* – parameter declared in the algorithm; value  $pf_c$  (probability of force creation) is calculated according to the value of force;

<sup>26</sup> The factor  $\sqrt{3}/2$  is applied to one of the orthogonal directions (the axis OY) because the lattice is triangular in fact

- $f_x$  - is calculated according to the simulation outputs *transfer14*, *transfer25*, *transfer36*. This parameter is explained in Chapter 5.4.
- Time – it is the total time period of the simulation.
- Steady state of the flow – it is the number of time steps after the averaging of the flow velocity is being started.

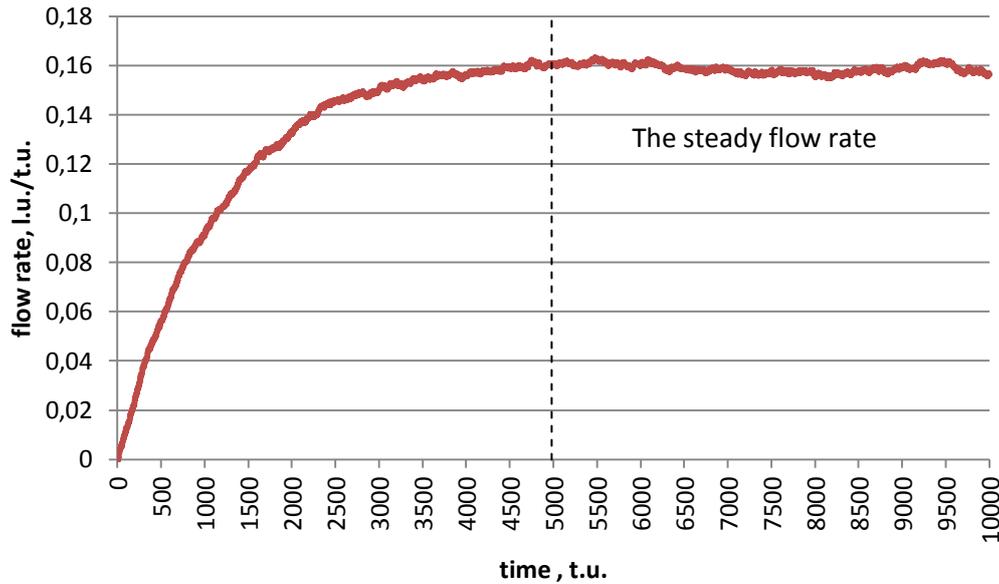
## 5.4. Results and discussion

According to the *Table 3* settings for *Poiseuille flow* computer are varied due to the channel width  $d$  and pressure gradient created using probability of force creation at the left boundary of the channel  $pf_c$ . The pressure gradient is imposed on the lattice by the parameter  $pf_c$  applied equally along the left boundary of the channel. The similar method was exploited for example in [79] and [83]. The pressure gradient is created here in terms of reversing particle momentum vectors. Reversing of particles is done by the certain probability. This process is applied for all nodes of one or more columns of the lattice (according to the width of the imaginary ventilator). The length of columns is equal to the channel width  $d$ . The width of the imaginary ventilator is two columns of lattice nodes due to the lattice geometry and is the same in all carried out simulations.

Probability of force creation is expressed below as  $f_x$ . To be more concrete the parameter  $f_x$  expresses the average change of the  $x$  component of the particle momentum at a particular node during one time step (i.e.  $1 t. u.$ ). From the *Figure 36* it is evident, that fluid flows in the channel to the right. The flipping mechanism impresses merely on particles with negative  $x$  components of velocity at the left side of the channel. The “total force” applied on the line of nodes is then  $nf_x$ , where  $n$  represents the number of nodes in the line that spans across the channel width. Thus, the pressure  $P$  applied at the left hand channel side is according to (86) and (91) the force per unit area and is expressed as a  $P = nf_x/d$ . Here the physical unit of  $P$  is  $(m. u.)/(t. u.)^2$ <sup>27</sup>, subsequently parameter  $f_x$  has unit  $(l. u.)/(t. u.)^2$ . When pressure gradient value is obtained, the “total force”  $nf_x$  is being divided by the product of the channel length and the channel width  $L \times d$ , then the unit of the pressure gradient is  $(m. u.)/(l. u.)(t. u.)^2$ .

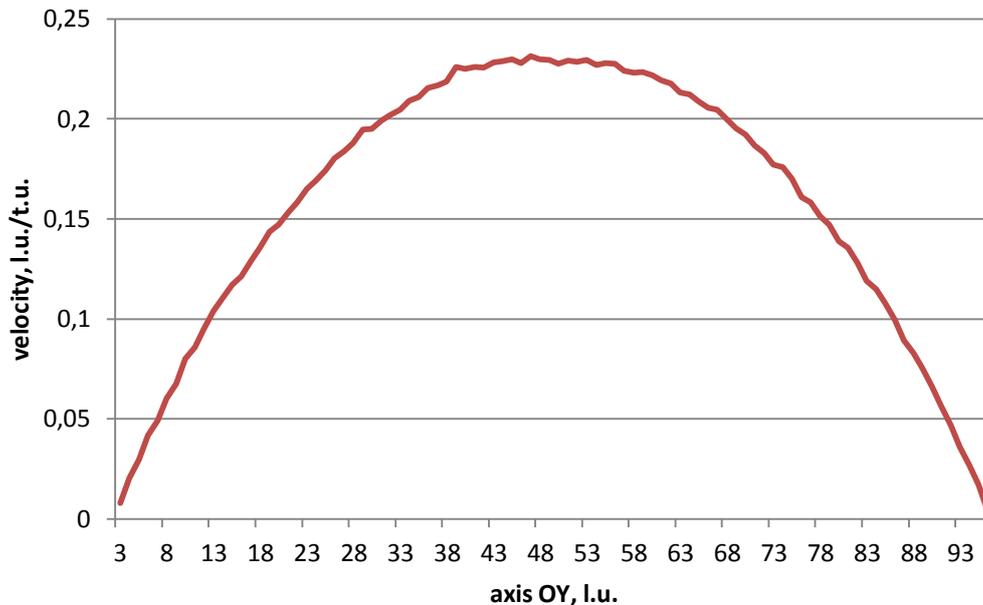
Results of the representative simulation are shown in *Figures 40* and *41*. The parameters of the channel width and flow are as follows: the width  $d = 100\sqrt{3}/2 l. u.$  and  $f_x = 0,3 l. u./t. u.^2$ ; these parameters are chosen for that example. *Figures 40* plots the flow rate as a function of time. Flow rate is computed by calculating the average  $x$ -component of velocity of all particles in the lattice. The steady flow rate is achieved approximately after  $5\ 000 t. u.$  The flow rate at the steady state is about  $0,16 l. u./t. u.$  (see *Figures 40*, the average value of the flow rate in the region “The steady flow rate”).

<sup>27</sup>  $P = \frac{F}{d} = \frac{ma}{d} = \frac{(m.u.)(l.u.)}{(l.u.)(t.u.)^2} = \frac{(m.u.)}{(t.u.)^2}$



**Figure 40:** The flow rate as a function of time for  $L = 550$  l.u.,  $d = 100\sqrt{3}/2$  l.u.,  $f_x = 0,3$ . The time period of the simulation measured in time units (t.u.) is given at the axis OX. Steady state of the flow is achieved after about 5000 t.u.

As it is evident from the *Figure 41* the parabolic shape of the flow velocity profile is obtained. The  $x$  component of velocity was averaged over the whole channel length  $L$  for each horizontal row of the lattice nodes over 5 000 t.u. in the steady state region of the flow in order to obtain velocity profile. These computer simulation outputs exhibit a parabolic velocity profile that is typical for a plane *Poiseuille* flow.

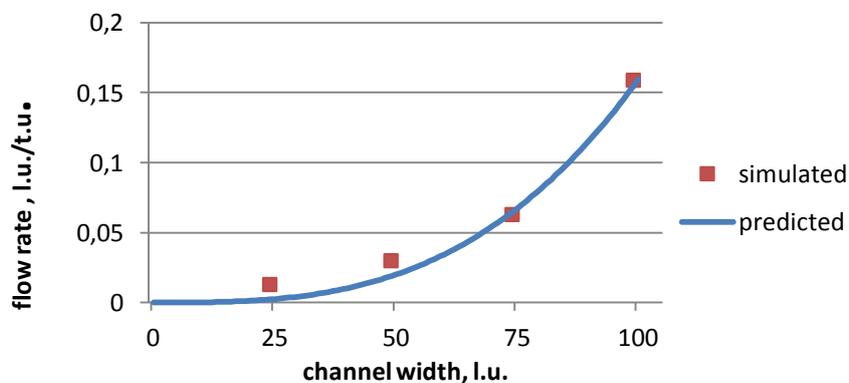


**Figure 41:** The velocity profile of the flow. Values of the  $x$  component of flow velocity averaged over the whole channel length (i.e. 550 l.u.) are at the axis OY. The vertical distance from the bottom wall of the channel named here as a “axis OY” and it is presented at the axis OX of the graph

Twenty independent experiments are carried out for  $f_x$  values of 2, 1.4, 0.4, 0.2 and 0.03  $l.u./t.u.^2$ . The width of the channel  $d$  is ranging from  $25\sqrt{3}/2$  to  $100\sqrt{3}/2$   $l.u.$  with the increment  $25\sqrt{3}/2$   $l.u.$  (see *Table 3*). Similar results are obtained for all realized computer simulations (see *Appendix J*). In order to evolve a steady state flow, the system was left to relax after the start of each simulation trial. The steady flow rate is achieved approximately after  $1000 \div 5\,000$   $t.u.$  according to the applied pressure gradient, that is being influenced by the parameter  $f_x$ , and the width of the channel  $d$ . The smaller the value  $f_x$  is applied, the longer time period is needed for an achievement of a steady state flow when the channel width  $d$  is constant. The smaller the width of the channel is, the shorter time period to reach the steady state of the flow it takes. Similarly to the representative results presented in *Figures 40* and *41*, all velocity profiles were averaged over the time in a steady state flow from  $t = 5000$   $t.u.$  to  $t = 10000$   $t.u.$

The influence of the pressure gradient and the channel's width on a shape of velocity profiles is evident and presented in *Appendix K*. If to compare fast and slow flow, the smoother shapes of velocity profiles can be observed in a faster flow. The higher the value of  $f_x$  is, the higher pressure gradient is applied on a channel. Subsequently, the higher value of  $f_x$  is, the faster the flow is in a channel – if we compare results obtained for the channel of the same width  $d$ .

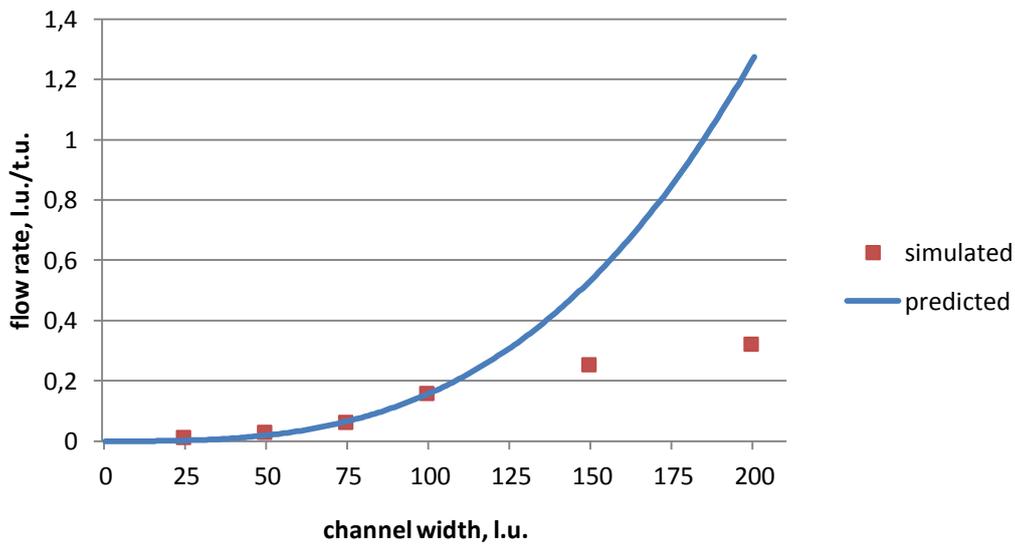
The relationship between channel width  $d$  and the flow rate measured as  $q = v_x$ , where  $v_x$  is the average  $x$  component of flow velocity per particle averaged over the entire lattice in a steady period of the flow is presented in *Appendix L*. Three representative examples are presented in that appendix: the fastest flow produced by the maximum pressure gradient (i.e.  $f_x = 2$   $l.u./t.u.^2$ ), the slowest one ( $f_x = 0,03$   $l.u./t.u.^2$ ) and the middle example ( $f_x = 0,4$   $l.u./t.u.^2$ ). Each figure contains the plot of observed values  $q$  (averaged over the time period  $t = 5000 \div 10000$   $t.u.$ ) as a function of the channel width  $d$  compared to the theoretical values of volumetric flow rate  $q$  predicted by (38). The viscosity  $\mu = 2,31$  for the theoretical curve was taken from *Rothman* [79], who simulated the lattice gas flow of the same density i.e.  $\rho = 2,5$  particles per node. The best matches between the theory and simulated results are presented in *Figure 42* and obtained for  $f_x = 0,4$   $l.u./t.u.^2$ .



**Figure 42:** Predicted and simulated volumetric flow rate as a function of channel width for a pressure gradient created using  $f_x = 0,4$  and the range of the channel width  $d=25\div 100$   $l.u.$

According to [79] good agreement between the theory and this type of computer simulation experiment are in coincidence with each other. Only when the width of the channel  $d$  is small (less than  $20 l.u.$  – it is not a case of the computer simulations presented in this chapter), or when both  $d$  and  $f_x$  are large, then the theory and computer simulation results disagree. In a case of large value of the channel width  $d$  the predicted flow rate is fasten (see *Figure 43*).

Two more computer simulations were implemented and evaluated for verification of above mentioned statement (i.e. for  $f_x = 0,4 l.u./t.u.^2$ ,  $d = 150\sqrt{3}/2 l.u.$  and  $200\sqrt{3}/2 l.u.$ ). An anomalously slow flow and its contraposition with predicted flow rate took place according to the limited range of possible velocities that Lattice Gas Cellular Automata is being able to simulate (see *Appendix L*,  $f_x = 2 l.u./t.u.^2$ ). The limited value of the flow rate is about  $0,2 l.u./t.u.$  The flow rates greater than that value are in contraposition with an equation expressing the *plane Poiseuille flow*, because it is too fast for the assumption of fluid incompressibility.



**Figure 43:** Predicted and simulated volumetric flow rate as a function of channel width for a pressure gradient created using  $f_x = 0,4$  and the range of the channel width  $d=25\div 200 l.u.$

Verification of the *Darcy's law* is presented in *Appendix M*. The linear dependence between flow rate in a steady state and pressure gradient is proved for all simulated channel widths  $d$ . Values of the pressure gradient for various flow rates are close to the line of linear regression. In graphs in *Appendix M* the relationship between geometry of the channel and pressure gradient is presented. If we consider the same length of the channel (it is  $550 l.u.$  in that series of simulations) and the same flow rate (for example  $q = 0,1 l.u./t.u.$ ), it is obvious the wider is the channel width the smaller is the pressure gradient.

Thus, proposed by FHP-1 Lattice Gas Cellular Automata algorithm model is able to simulate fluid flow between two parallel plates with periodic boundary conditions. Results, obtained from twenty experiments had proven the parabolic velocity profile of the flow and the

*Darcy's law*. Those simulation outputs are in a good agreement with results obtained by *Rothman* [79]. The range of admissible dimensions of the space for fluid flow simulation is obtained. First, the pressure gradient and the geometry of the porous media must be chosen in accordance to the limit of the flow rate ( $q = 0,2 \text{ l.u./t.u.}$ ). It is not recommended to use maximum probability of force creation ( $f_x = 2 \text{ l.u./t.u.}^2$ ) when the *plane Poiseuille flow* is simulated. An appropriate range of channel widths for that type of flow is obtained ( $d = 25 \div 100 \text{ l.u.}$ ).

Models of fluid flow in porous media under different conditions could be designed with the same basic approach outlined in Chapter 5.

## 6. COMPUTER SIMULATION OF THE TWO-DIMENSIONAL FLUID FLOW THROUGH POROUS STRUCTURES

In previous chapters Lattice Gas Cellular Automata were described from their essence. The developing of the own LGCA model for fluid flow simulation, its verification under different conditions and comparison between particular theoretical assumptions and results obtained by means of computer simulations was performed. As it was described in Chapter 4 Lattice Gas Cellular Automata model is able to describe fluid movement at its molecular level. It can be also expose that the individual particles moving, in a study of diffusion phenomenon for example, can be studied using LGCA model. Experiments presented in the Chapter 5 have proved that the same model using the same algorithm can describe also the fluid flow and finally it can substitute the hydrodynamic equation including *Navier-Stokes equations*.

In this chapter I will try to verify the particular hypothesis related to the curious behaviour of the fluid flow that was not proved yet. Let consider the filtration through assembled filter – i.e. filter consists of many pleats. What directions the fluid flows inside the assembled filter? In order to answer the question the developed FHP-1 Lattice Gas Cellular Automata model is used as the numerical and visualization technique.

### 6.1. Theoretical assumption

Filtration is defined as a mechanical or physical operation used for the separation of solids particles from fluid ones. Filtration is based on a fluid flow phenomenon, when fluid flows from the high to the low pressure side of filter leaving some material behind. Nowadays, many types of filters exist. There are, for example, granular filters, membrane filters and filters based on fibrous materials.

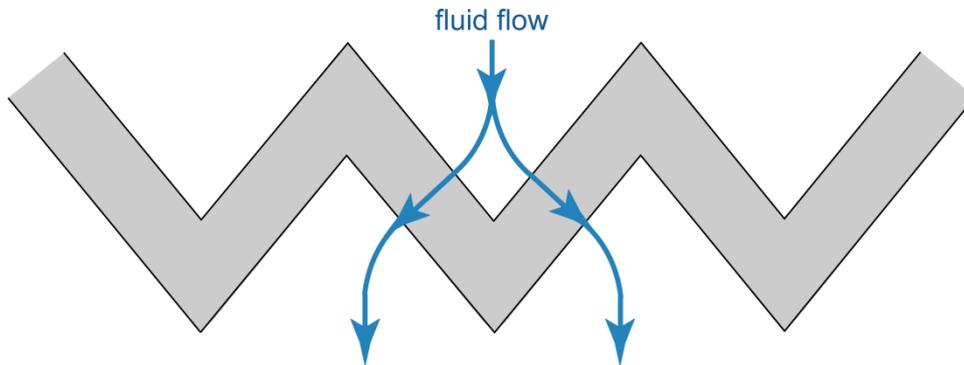
According to [84] relationship between filtration characteristics and geometry of the porous structure is given by *Darcy's law*, which is valid for the laminar regime of fluid flow. In contrast to Equation (39), the *Darcy's law* includes geometric characteristics of filter:

$$q = \frac{Ak dp}{h\mu dx} \quad (40)$$

where  $A$  is a filtration area and  $h$  is a filter thickness. The internal structure of porous medium is given by the permeability coefficient  $k$ . From the Equation (40) it is obvious, that pressure gradient is linearly dependent on a filtration area.

Filters based on a pleated porous material are required because of their high efficiency, durability and low pressure drop. These properties are obtained because of several times bigger filtration area, which decreases the pressure gradient. *Brown* in his work [85] has hypothesized that the good filtration characteristics of assembled filters are obtained because the specific orientation of the fluid flow inside the pleats of filters. *Brown* explains his assumption as following: "...the special profile of the fluid velocity field is given by the

minimization of kinetic energy dissipation due to the viscous friction". In other words, fluid moves the path towards the least resistance. For that reason, the flow of the filtered dispersion tries to orient itself perpendicularly to the filter area in order to minimize the distance, which has to be pass in a side of porous matter (see *Figure 44*).



**Figure 44:** Theoretical flow pattern through pleats at assembled filter

*Hrůza* [84] has studied the filtration characteristics of assembled filters, produced from nonwoven materials (spunbond or/and meltblown). Some experiments were focused on using nanofibre layers in such type of filters. He has obtained the results that confirm the Brown's idea, but do not demonstrate the fluid path through a assembled filters.

Unfortunately, no visual proofs of the phenomenon were found. Only theoretical assumption, presented by *Brown*, was found in literature. Thus, the aim of the computer simulation proposed by me is to prove the convolution of the flow direction at the boundary with the random porous media imitating the structure of nonwoven textile.

## 6.2. FHP-1 Lattice Gas Cellular Automata algorithm for fluid flow through porous medium simulation

The FHP-1 LGCA algorithm for Poiseuille flow simulation described in detail in the Chapter 5.2, is used for a simulation of the fluid flow through a porous medium. The newly developed parts of the algorithm are described in this chapter. The full code of the algorithm is presented in the *Appendix N*. Supplementary algorithms, which are used for averaging and graphical representation of output data, are presented in the *Appendix O*.

### 6.2.1. Code fragment 1 – Header files and initialization of the simulation domain

Compared to the FHP-1 LGCA algorithm, developed in accordance to Poiseuille flow simulation, special variables and parameters are declared in this part of the algorithm:

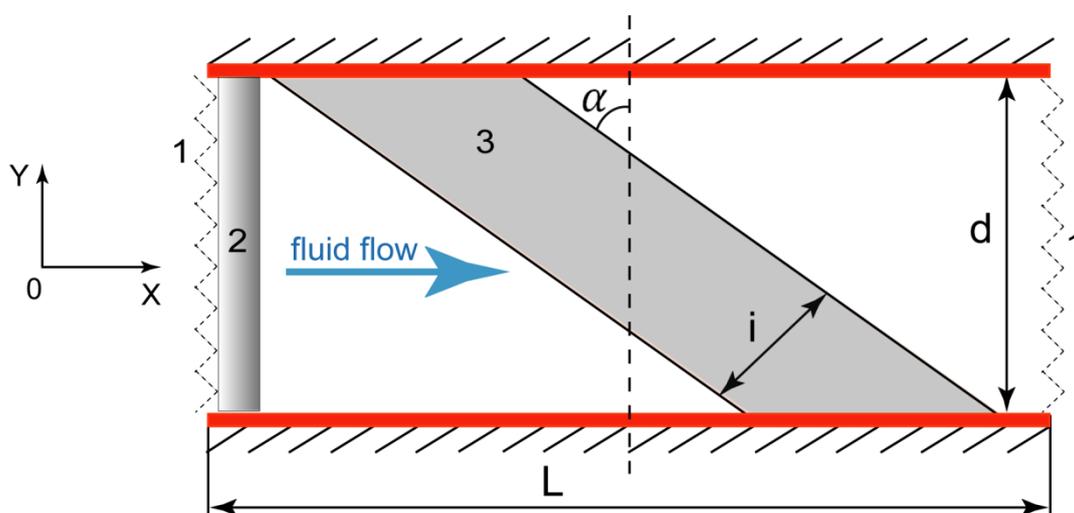
- *velfieldx* and *velfieldy* corresponds to the *x* and *y* components of velocity vectors. They are calculated in all lattice nodes, where moving particles are occurred.

- $\alpha$ ,  $b_1$  and  $b_2$  are used for porous medium generation.
- $\alpha$  is a parameter that determines the angle, at which porous medium crosses the vertical channel axis.
- $\text{porousmedium}$  is a probability of moveless particles generation in a position of porous medium
- $i$  – determines the one half of the porous medium thickness. Hence, the width of the porous medium is  $2i$ .
- $\text{pore}$  and  $\text{fibre}$  correspond to the number of empty lattice nodes and moveless particles respectively calculated in a position of porous medium.
- $\text{fluid}$ ,  $\text{obstacle}$ ,  $\text{hole}$  are parameters, which represent a colour of moving particles, moveless ones and a colour of empty lattice nodes at a graphical output.

One more subroutine has to be declared compare to the FHP-1 LGCA algorithm, designed for Poiseuille flow simulation. It is  $\text{velocityfield}()$  which calculates  $x$  and  $y$  components of the moving particles velocity.

### 6.2.2. Code fragment 3 – Creation of the simulation domain

The channel with upper and bottom solid boundaries is created (see *Figure 45*). The length of the channel is  $L = 450 \text{ l.u.}$ , the width  $d = 250\sqrt{3}/2 \text{ l.u.}$ . First, the random porous structure is generated in a whole area of the simulation domain. Frequency of moveless particles occurrence is controlled by the parameter  $\text{porousmedium}$ . Boundaries of the porous medium are determined according to parameters  $\alpha$  and thickness of the porous medium  $i$  then. All lattice nodes behind lines  $b_1$  and  $b_2$  are kept at their original value, i.e. zero. As a result, porous medium of a certain thickness and porosity is generated inside the channel at a certain angle towards to the vertical channel axis.



**Figure 45:** The geometry of two-dimensional channel for fluid flow through porous medium simulation:  $L$  is the length and  $d$  is the width of the channel,  $\alpha$  is an inclination angle of the porous medium,  $i$  is a one half of the porous medium thickness, 1 – periodic boundary conditions, 2 – the imaginary ventilator. The vertical dot line presents the vertical axis of the channel

### 6.2.3. Code fragment 6 – The main cycle of the algorithm

The main cycle of the algorithm has the same structure as it was presented in the previous algorithm (see Chapter 5.2.4, *Figure 37*). In contrast to previous algorithm the calculation of the  $x$ - and  $y$ - components of particles velocity occurs in a steady state region of the flow. This operation is described in details in following Chapter 6.2.4.

### 6.2.4. Code fragment 9-B – Distribution of velocity vectors of moving particles

After the steady state of the flow is achieved,  $x$  and  $y$  components of a total velocity vector in each lattice node starts. The subprogram *velocityfield()* is being used for this reason. It is working according to following steps:

1. Selecting the lattice node with coordinates  $x$  and  $y$  (except lattice nodes occupied by solid moveless particles).
2. Calculation of the  $x$  component of the total velocity *velfieldx* in the lattice node.
3. Calculation of the  $y$  component of the total velocity *velfielddy* in the lattice node.
4. Repeating previous steps for all lattice nodes systematically.

This calculation is carried out at every time step in a steady state of the fluid flow. This approach allows to obtain a set of random states of the simulated system. Subsequently, application of averaging over many random states provides more accurate estimation of mean values.

Presented computer simulation allows to observe the distribution of velocity vectors. The length of each observed vector corresponds to the time and space-averaged velocity of moving particles in a node inside the simulation domain. Space-averaging is performed using the first of supplementary algorithms in the direction of *Appendix O*. Velocities of particles are space-averaged inside the  $5l.u. \times 5l.u.$  squares. The last supplementary algorithm is used for a graphical representation of data averaged in space and time. For every lattice node the length of the velocity vector is calculated according to the *Pythagorean theorem*, where sides of the right triangle are  $x$  and  $y$  components of the velocity vector.

## 6.3. Simulation setup

The two-dimensional channel geometry is employed for a chosen set of computer simulation experiments. Overhead and bottom channel sides are composed of moveless particles which imitated channel's walls, similarly to the simulation of fluid flow in a channel. The length of the channel  $L$  is 450 lattice units ( $l.u.$ ) and due to the usage of the periodic boundary conditions the infinitely long channel in  $x$  direction is in fact created. The width of the channel  $d$  is  $250\sqrt{3/2} l.u.$

The design of the assembled filter is slightly simplified: fluid has flown only through the one part of a filter pleat. For that reason, porous medium is placed in the middle part of the channel at the defined angle. Its thickness  $d$  is 90 l. u. Generated fluid particles are directed into the free space of the simulation domain as well as between moveless particles of solid walls and porous medium. In order to attend viscous flow (i.e. three particles collisions) the average density  $\rho$  of 3 particles per node is used in each simulation. The bounce-back type of reflective boundary conditions is pre-set for the fluid particle collisions with moveless particles at channel walls as well as for fluid particle collisions with moveless particles of the porous material. Pressure gradient is created in a same way as it was described in previous computer simulation (see Chapter 5.1). The exact simulation setups are presented in the Table 34.

Table 4: The list of fluid flow through porous medium computer simulations and their setups

	Size of the channel, l. u.		average density, m.u./l.u.	force / pfc <sup>28</sup>	Param eter $f_x$ <sup>29</sup>	Porosity of the porous medium	$\alpha$	Time of the simulatio n, t. u.
	The length	The width						
1.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,95	15°	10000
2.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,9	15°	10000
3.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,85	15°	10000
4.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,7	15°	10000
5.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,95	35°	10000
6.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,9	35°	10000
7.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,85	35°	10000
8.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,7	35°	10000
9.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,95	55°	10000
10.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,9	55°	10000
11.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,85	55°	10000
12.	450	$250\sqrt{3}/2$	2 / 3	30 / 0,3	0,6	0,7	55°	10000

This table includes value of the the following parameters:

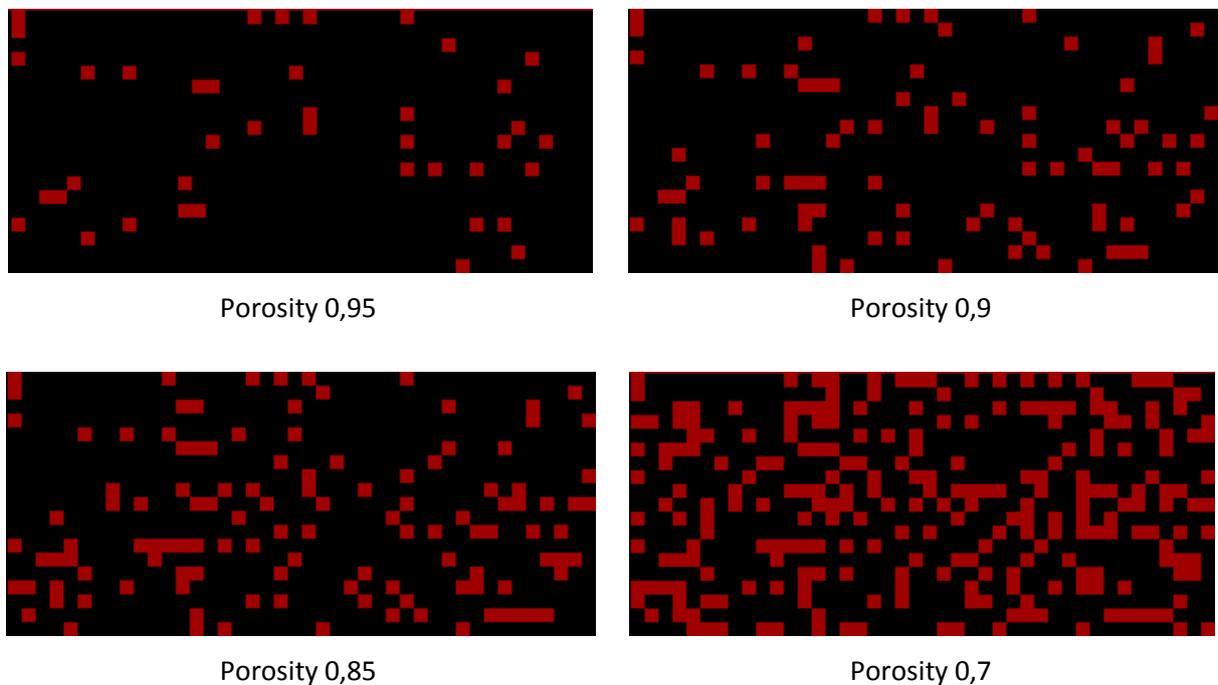
- Size of the channel - it is presented by its length  $L$  and width  $d$ ;
- Average density – corresponds to the average number of moving particles in the lattice node;
- *force* – parameter declared in the algorithm; value *pfc* (probability of force creation) is calculated according to the value of force;
- $f_x$  - is calculated according to the simulation outputs *transfer14*, *transfer25*, *transfer36*. This parameter was explained in details in the Chapter 5.4.

<sup>28</sup> Parameter *force* was declared in the algorithm; value *pfc* (probability of force creation) is calculated.

<sup>29</sup> Parameter  $f_x$  was calculated according to the simulations outputs *transfer14*, *transfer25*, *transfer36*

- Porosity of the porous medium – is calculated as a ratio between the number of *pore* nodes and sum of *pore* and *fibre* nodes. Parameters *pore* and *fibre* are declared in the algorithm;
- $\alpha$  – it is an inclination angle of the porous medium relative to the axis OY of the channel;
- Time of the simulation – it is the total time period of the simulation.

It is evident from the *Table 3* that settings for these computer simulations vary in the value of porosity of the porous medium and the angle, at which the porous medium crossed the vertical channel's axis. Porosity values, chosen for this experiment, correspond to real porosities of nonwoven materials (i.e. 0,85 – 0,95). Porosity 0,7 approximates the porosity of nanofibre layers (porosity of nanofibre layers ranging between 0,5 and 0,85). Simulated structures of porous media are shown in *Figure 46*.



**Figure 46:** Random structures of porous media generated in the computer simulation experiment. Porosity ranging from 0,7 to 0,95

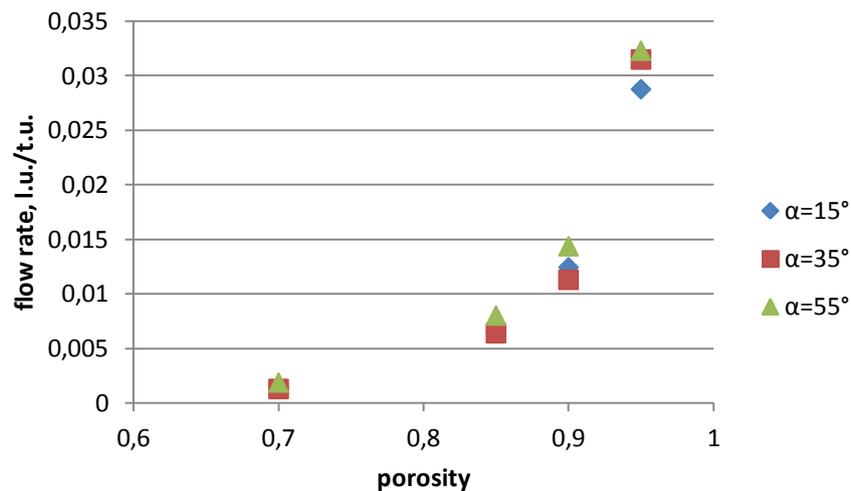
## 6.4. Results and discussion

Twelve independent computer simulations were performed according to the *Table 4*. The results of those simulations are obtained by specifying the parameter  $f_x = 0,6$  which expresses an average change of the  $x$  component of the particle momentum at a particular node during one time step. Detailed explanation of the parameter  $f_x$  was presented in Chapter 5.4. Time evolution of the FHP-1 LGCA model for fluid flow in porous medium modelled with a reduced simulation domain ( $L = 48 l.u.$ ,  $d = 25\sqrt{3}/2 l.u.$ ) is introduced in *Appendix P*. The low probability of channels occupation is deliberately chosen for verification

of particles collisions and periodic boundary conditions. The average density is 0,2 particles per a lattice node here. Simulated system is monitored for  $t = 20t.u.$

First, all simulated systems are left to achieve the steady state. The steady states of the fluid flow are achieved after about  $2000 \div 7000 t.u.$  (see *Appendix Q*). Achieving of a steady flow inside the reduced simulation domain is obvious from the *Appendix R*. Simulation domain of reduced size, where the length of the channel  $L = 48 l.u.$  and the width  $d = 25\sqrt{3}/2 l.u.$  are used. The average density is equal to 3 particles per lattice node. System's configuration are recorded after every ten time steps for  $t = 150 t.u.$ , i.e. when the steady state of the flow is reached.

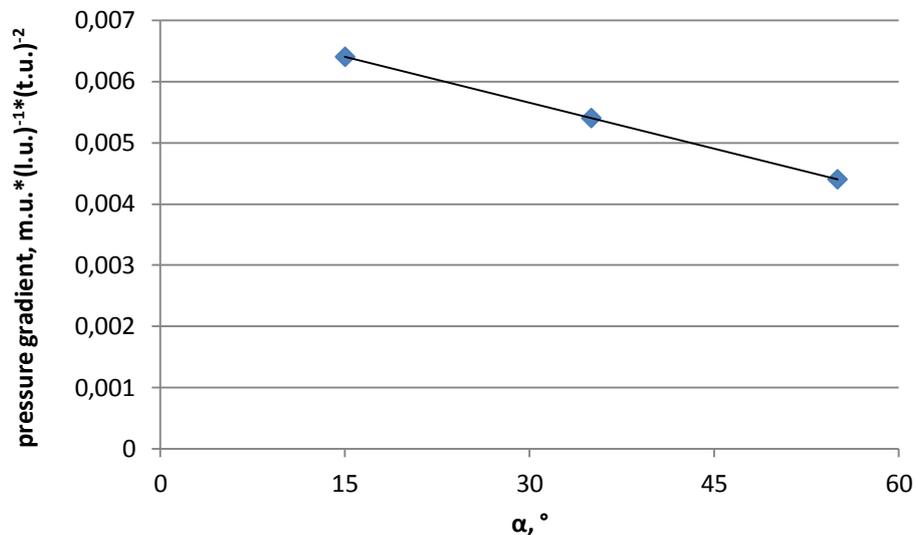
In real simulations the smaller value of the porosity the system has and the higher inclination angle of the porous medium (i.e  $\alpha$ , see *Figure 45*) is simulated (i.e. the biggest surface area of the porous medium), the longer time it takes to reach the steady flow. Flow rates calculated in steady states as a function of the porosity and the inclination of porous medium are presented in *Figure 47*. The increasing of both the porosity and fluid flow rate is obvious from this figure.



**Figure 47:** Fluid flow rate as a function of porosity and inclination of porous medium for pressure gradient created using  $f_x = 0,6$

From the Darcy's law (see Equation (40)), fluid velocity is linearly dependant on pressure gradient applied at porous medium. It is obvious from the Chapter 5.4 that pressure gradient in this type of simulation models is directly dependant on value of parameter  $f_x$  and decreases with surface area to which it is applied. Because  $f_x$  is constant in all simulation experiments, the pressure gradient is mainly influenced by the inclination of the porous medium. With increasing  $\alpha$ , surface area of the porous medium increased too. Relationship between the inclination angle  $\alpha$  and resulting pressure gradient is presented in *Figure 48*. This relationship was introduced by *Brown* [85] and experimentally verified by *Hrůza* [84]. According to *Brown*, the pleating of assembled filters increases the area of the material that can be accommodated in a fixed volume and so it reduces the filtration velocity. Therefore,

the pressure drop at fixed volume flow is reduced too. That is why the pressure drop decreases as the number of pleats per unit length grows” [85].



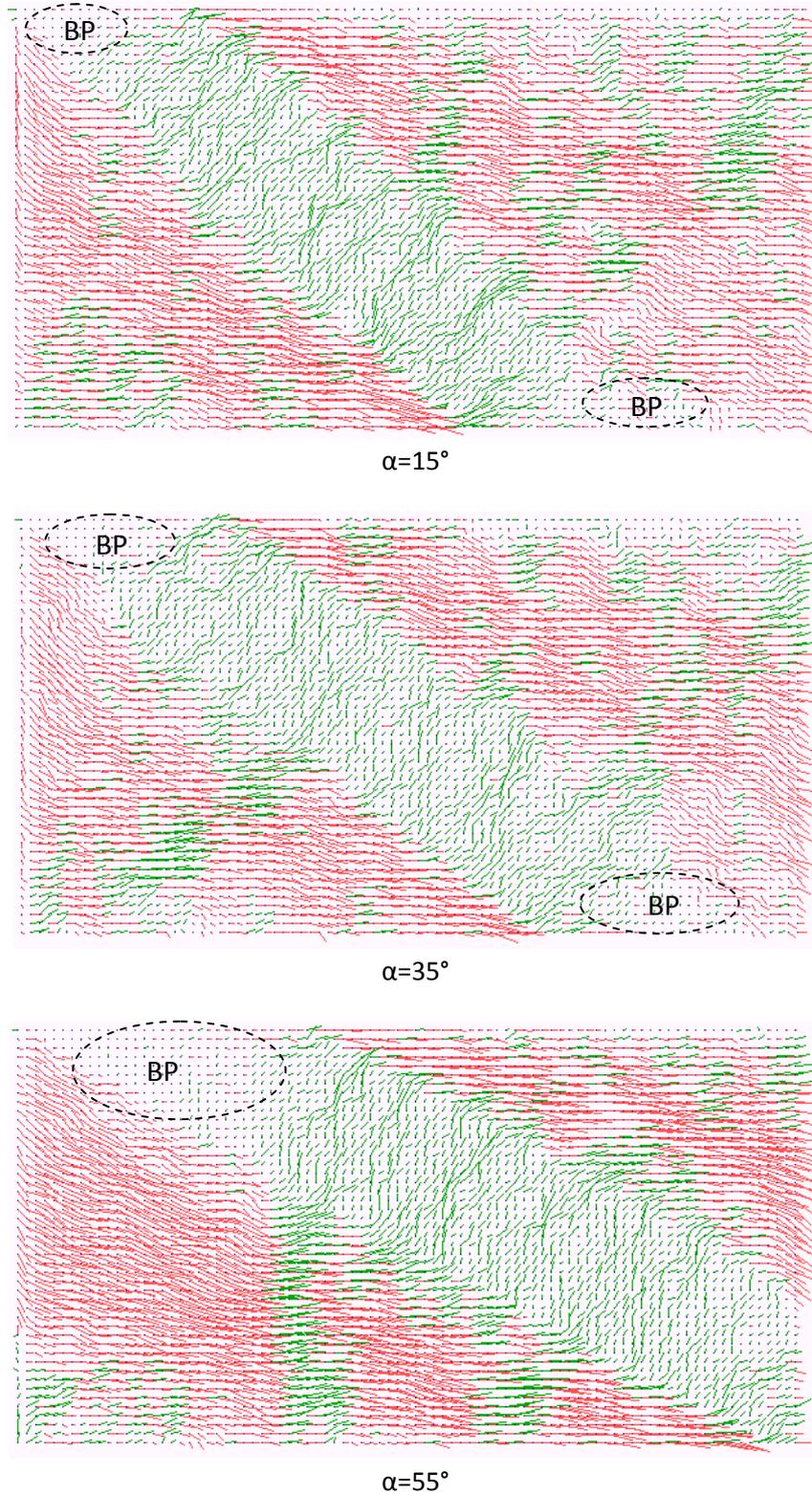
**Figure 48:** Pressure gradient created using  $f_x = 0,6$  as a function of inclination angle  $\alpha$  indicates the orientation of the porous medium in a channel

Velocity fields are monitored and expressed by graphical manner for all setups of the simulated system for better understanding of the phenomenon which was introduced in Chapter 6.1. Velocity vectors are obtained by averaging over  $5 l.u. \times 5 l.u.$  squares on the lattice and over  $8000 \div 3000 t.u.$  of the steady flow state. In this way the velocity vector arrays are obtained.

Several unique features of flow through a porous medium are illustrated in *Figure 49*. For better realization of the velocity vectors two colours are used. If the velocity vector points in a first or second quadrants (i.e. it is from the interval  $(0, \pi]$ ), then it obtains green colour, other way, it turns into red. It is evident that on the interface between the free channel area and the porous structure appears a reorganization of fluid velocity directions. The flow makes an impact on a solid parts of the porous medium, thus fluid particles do try to stream to the pores inside the porous material. It is possible to see (see *Figure 49*), that the fluid enters into the porous material perpendicularly. The same results are obtained when the inclination angle  $\alpha$  of porous medium is  $15^\circ$ ,  $35^\circ$  and  $55^\circ$  and the porosity is 0,95 or 0,9 or 0,85. Some regions of the porous medium was relatively stagnant. The local fluid flow in “blind pores” close to channels walls is zero.

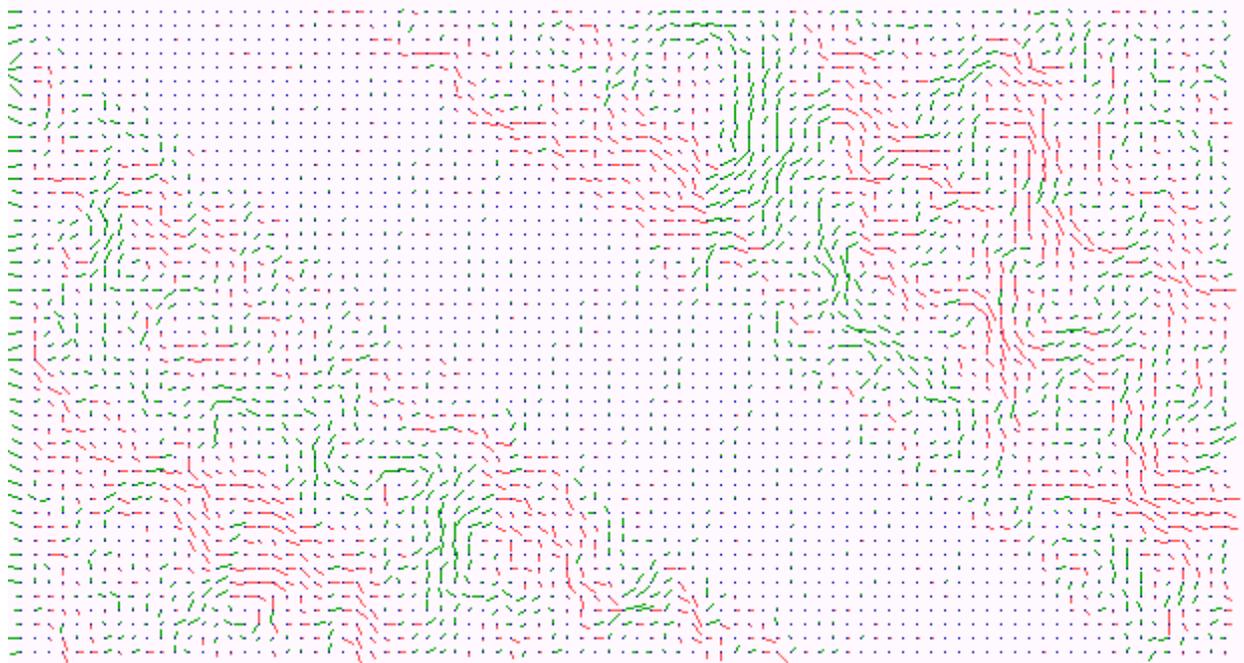
An interesting behaviour of the flow is monitored for porous structure with porosity 0,7 (see *Figure 50*). The same uniform body force at the left boundary of the channel ( $f_x = 0,6$ ) is created, but local velocity vectors are smaller compared to three previous results. Stagnant area covers here the whole space of the porous medium. The local fluid flow in such a dense porous structure is close to zero. It is obvious from previous results, that flow rate for this

value of porosity is almost zero. Winding paths are evident in front of, and behind the porous medium. Circulating eddies are evident in those parts of the channel.



**Figure 49:** Fluid velocity directions inside the declined porous material with random structure for porosity 0,95 and  $\alpha=15^\circ$ ,  $35^\circ$  and  $55^\circ$ . Region BP corresponds to “blind pores” of the porous medium

The same results, as described in this chapter, were obtained for the whole range of porosity and inclination angle  $\alpha$  introduced in the Table 4 (see *Appendix S*). These computer simulation results are qualitative only, but they show the nature of the phenomenon in question.



**Figure 50:** *Fluid velocity directions inside the channel and declined porous material with random structure for porosity 0,7 and  $\alpha= 35^\circ$*

The reorganization of fluid flow inside the declined porous structure using the designed FHP-1 LGCA model was proved. Results obtained in this computer simulation sets are in a good agreements with results obtained by *Hruža* [84] and conform the hypothesis of *Brown* [85].

## CONCLUSIONS

In a frame of this work a two-dimensional non-deterministic Lattice Gas Cellular Automata algorithm based on the FHP-1 LGCA model was developed and described in detail. Algorithm was created in a C++ programming language, Borland version 4.0. Basic skeleton of the algorithm and function of its particular code fragments were minutely described. The full text of the algorithm including all technical aspects was introduced in the appendix part of the thesis. The basic Lattice Gas Cellular Automata algorithm has an universal structure and was easily modified for various versions. Phases of the LGCA evolution process (collision and propagation) take place in subroutines. If boundary conditions are changed, it does not interfere into the main part of the algorithm. Due to adaptation of the hexagonal lattice to the square one and using different arrangement of neighbourhood in odd and even rows of the lattice calculations were more complicated. On the other hand, this approach allows productively to utilize all points of the simulation domain. The main feature of the algorithm is its non-deterministic evolution in time. During the collision phase new state of a finite automaton is always generated randomly according to the conservation of mass and momentum in the lattice node. No predefined matrix of states changes was used. This property allows to model the fluid flow in more realistic way.

The two-dimensional Lattice Gas Cellular Automata algorithm, developed in a frame of this work, was verified using two independent tests. First of them, Brownian motion, simulates steady flow and was aimed on a monitoring of the one moving particle among many other fluid particles. Brownian motion was simulated inside the simulation domain of the size  $300 \times 300$  or  $400 \times 400$  *lattice units* for the period of time  $t = 10000$  *time units*. By mean of this test the set of Brownian particle's paths was obtained. It was noted that paths walked by the Brownian particle are far from linear. Many movements round and round or returning back to the starting point were monitored. There is a linear relationship between the mean square distance of the Brownian particle and time according to theoretical assumption. Based on two computer simulation experiments, varied in density of lattice gas, the theoretical assumption was proved. It can be argued that algorithm is working in a right way according to the results obtained in that test. Simulated system exhibits behaviour close to the real one. To limit the degree of data fluctuation around the linear regression the usage of biggest size of the simulation domain and extension of the simulation period were suggested. This simulation can be used not only for the verification of a newly developed algorithm. It allows the study of diffusion phenomena including calculation of the diffusion coefficient. Another possible usage of the created algorithm is modelling of polymer molecules shapes.

The Lattice Gas Cellular Automata algorithms, developed for fluid flow modelling, are predominantly verified by means of Poiseuille flow simulation. It was noticed, Poiseuille flow simulation is the most popular benchmark test in a case of a fluid flow study. The special Lattice Gas Cellular Automata algorithm based on the FHP-1 LGCA model was designed for that reason and described in details. The algorithm supposes the simulation of

incompressible fluid flow between two stationary parallel plates driven by constant body force. The bounce-back type of free reflections was used along the walls of a channel and periodic boundary conditions were applied at the both vertical boundaries of the channel. Behaviour of the simulated system was studied under various simulation setups. The main aim of the test was obtaining the parabolic profile of the flow. Settings for Poiseuille flow computer simulations were varied due to the channel width and pressure gradient created using certain probability of force creation at the left boundary of the channel. So, twenty different parabolic profiles of flow velocity were obtained. The slower fluid flow was simulated, the smoother velocity profile was obtained. The channel width had a little effect on a shape of the velocity profile. The smaller the channel width was simulated, the more peaked velocity profile was obtained. From physical point of view, correctness of the developed LGCA algorithm for fluid flow simulation is noticeable not only from the shape of velocity profiles but also from the graphs, where the predicted and simulated relationship between flow rate and channel width were compared. Good agreement between prediction and simulation results was here observed for a range of channel width  $25 \div 100$  *lattice units*, and flows created by the gradient as a result of  $f_x = 0,4 \div 0,03$ . These results provide information about the appropriate settings of future simulations. Furthermore, based on a computer simulation outputs the Darcy's law was verified. The linear dependence between flow rate in a steady state and pressure gradient was proved for all simulated channel width. Finally, all outputs of the Poiseuille flow simulation are in a good agreement with the *Rothman's* simulation experiment.

Computer simulation of a physical phenomenon, the existence of which has not been demonstrated experimentally using accessible visualization techniques, was presented in the last chapter of the thesis. It concerned to the fluid flow through assembled filters. The theoretical assumption that special orientation of the fluid inside those filters leads to the good filtration characteristics of them was founded in literature [85]. Relationship between filtration characteristics and geometry of the internal structure of the filter was empirically obtained by *Hrůza* [84]. But his experiments did not prove the convolution of the flow direction at the boundary with porous media. Developed LGCA algorithm was modified for that study. Based on twelve computer simulations the reorganization of the fluid flow inside declined porous structure was obtained for different simulation setups. Influence of the inclination angle and porosity of the porous medium was studied. The results obtained from the computer simulation have shown, that proposed LGCA algorithm is suitable for a theoretical prediction of a fluid flow inside porous structures and also it can be used as a visualization tool.

Based on a study that was done in a frame of this work, the suitability of Lattice Gas Cellular Automata approach for fluid flow in porous structure modelling was demonstrated. The fluid flow in difficult multilayer textile structure is possible to study and visualize using modern microscopic techniques and developed LGCA algorithm. However there are a number of limitations in the current study. In a case of real porous media study is very difficult to

implement “good” boundary conditions for curved walls – always some degree of approximation should be here because of using the regular type of lattice.

### **Future work**

I would like to aim my future work toward those directions:

- (i) There is a need to calibrate the developed LGCA model and to determine the physical units of the simulation system for the fluid flow in porous media study. The algorithm for Brownian motion simulation can be used for calibration. It is possible to calibrate length unit of the system based on knowledge of a mean free path of lattice gas particle [92].
- (ii) The medical applications of nanofiber materials become topical with technical progress, development and production of nanoporous structures. Monte Carlo models and especially Lattice Boltzmann model begin to be popular. They are used as a simulation tool for the study of cell proliferation in scaffolds or flow in 3D porous scaffold materials.

## REFERENCES

- [1] **Hartmann, Stephan.** The World as a Process: Simulations in the Natural and Social Sciences. [book auth.] Rainer et al. (eds.) Hegselmann. Modeling and Simulation in the Social Sciences from the Philosophy of Science Point of View, Theory and Decision Library. Dordrecht : Kluwer, 1996, pp. 77-100.
- [2] **Stevenson, Timothy James.** Simulation of the vehicle-pedestrian interaction. Ph.D thesis: University of Canterbury, Mechanical Engineering, 2006. pp. 44-45.
- [3] **Nance, Richard E. and Sargent, Robert G.** Perspectives on the evolution of simulation. Operations Research. January-February 2002, Vol. 50, Issue 1, pp. 161-172.
- [4] **Rice, Stephen V.; Marjanski, Ana; Markowitz, Harry M.; Bailey, Stephen M.** Object-Oriented SIMSCRIPT. Proceedings of the 37th Annual Simulation Symposium. Arlington, Virginia, 2004, pp. 178-186.
- [5] **Buxton, John N. a Laski, John G.** Control and Simulation Language. The Computer Journal. 5, 1962, Vol. 3, pp 194-199.
- [6] **Holmevik, Jan R.** Compiling SIMULA: A Historical Study of Technological Genesis. IEEE Annals Of the History of Computing. 16, 1994, Vol. 4, pp 25-37.
- [7] **Reitman, Julian.** [ed.] Abrams, M., Haigh, P. and Comfort, J. Keynote Address: A concise history of the ups and downs of simulation. Proceedings of the Winter Simulation Conference. 1988.
- [8] **Shinde, Sagar.** Introduction to Modeling and Simulation: Historical Perspective. Simulation & Modeling Team. [Online] [Cited: 01.08.2008] <http://www.uh.edu/~lcr3600/simulation/historical.html>.
- [9] **Maria, Anu.** [ed.] Andradóttir, S. et al. Introduction to modeling and simulation. Proceedings of the 29th conference on Winter simulation. Atlanta, Georgia, United States: IEEE Computer Society, 1997. pp. 7-13. ISBN: 0-7803-4278-X.
- [10] **Fritzson, Peter.** Principles of Objective-Oriented Modeling and Simulation with Modelica 2.1. : Wiley-IEEE Press, 2004. pp. 3-18. ISBN 0-471-471631.
- [11] Stochastic process. Wikipedia – The Free Encyclopedia. [Online] [Cited: 03.11.2008] <http://en.wikipedia.org/wiki/Stochastic>.
- [12] **Jelen, J.** Determinismus a nahoda. [book auth.] Adamova, L., Dudak, V., Jelen, J. Kapitoly z filosofie vědy. Praha: ČVUT, 1993.
- [13] **Frigg, Roman and Hartmann, Stephan.** Models in Science. Stanford Encyclopedia of Philosophy. [Online] [Cited: 20.08.2008] <http://plato.stanford.edu/entries/models-science/>.
- [14] **Humphreys, Paul.** Numerical Experimentation. [book auth.] Suppes, Patrick and Humphreys, Paul. Patrick Suppes: Scientific Philosopher. Dordrecht: Kluwer Academic

Publishers, 1994, Vols. 2. Philosophy of Physics, Theory Structure, and Measurement Theory, pp. 103-121.

[15] Wikipedia - the Free Encyclopedia. Physical body. [Online] May 5, 2009. [Cited: May 9, 2009.] [http://en.wikipedia.org/wiki/Physical\\_body](http://en.wikipedia.org/wiki/Physical_body).

[16] **Zeng, X., et al.** Computational Textile: Hardcover, 2007. ISBN: 978-3-540-70656-4.

[17] **Lukas, David and Ocheretna, Larysa.** The cellular automata lattice gas approach for fluid flows in porous media. [ed.] N. Pan and P. Gibson. Thermal and moisture transport in fibrous materials. Cambridge: Woohnead Publishing Limited, 2006, pp. 357-401.

[18] **Leisen, Johannes, Beckham, Haskell and Farber, Peter.** Micro-Flow in Textiles: National Textile Center, 2005. Annual Report of NTC Project No. F04-GT05.

[19] **Halasová, Andrea.** Příspěvek k hodnocení prodyšnosti oděvních sendvičů v podmínkách rychle proudícího vzduchu. Liberec: Textilní fakulta, Technická univerzita v Liberci, 2007. 8/2007.

[20] **Roy, Subrata; Raju, Reni; Chuang, F. Helen; Cruden, A. Brett; Meyyappan, M.** Modeling gas flow through microchannels and nanopores. Journal of Applied Physics. Vol. 8, Issue 93, 2003, pp. 4870-4879.

[21] **Truesdell, C. a Muncaster, R.G.** Fundamentals of Maxwell's Kinetic Theory of a Simple Monatomic Gas. New York: Academic Press, 1980. ISBN 0127013504.

[22] **Brdička, Miroslav; Samek, Ladislav; Sopko, Bruno.** Mechanika kontinua. Praha: ACADEMIA, 2005. pp. 610-621. ISBN 80-200-1344-X.

[23] **Ландау, Л.Д.; Лифшиц, Е.М.** Гидродинамика. Москва: Наука, 1988. pp. 71-79. Vol. том VI.

[24] **Feynman, Richard P.; Leighton, Robert B; Sands, Matthew.** Feynmanovy přednášky z fyziky s řešenými příklady: FRAGMENT, 2001. pp. 740-746. Vol. 2/3. ISBN 80-7200-420-4.

[25] Clay Mathematics Institute. The Millennium Prize Problems. [Online] [Cited: 05.08.2010] <http://www.claymath.org/millennium/>.

[26] **Киттель, Чарльз.** Статистическая термодинамика. Москва : Наука, 1977. pp. 330-331.

[27] **Fredkin, Edward.** Digital Mechanics - An informational process based on reversible universal cellular automata. Physica D: Nonlinear Phenomena. 1990, Issues 1-3, 45. pp. 254-270.

[28] **Hillis, Daniel W.** Richard Feynman and the Connection Machine. Physics Today, 1989, pp. 78-83.

[29] **Wolfram, Stephen.** Statistical mechanics of cellular automata. Reviews of Modern Physics, Vol. 55, Issue 3, 1983, pp. 601-644.

- [30] **Тоффоли, Т. а Н., Марголус.** Машины клеточных автоматов. Москва: Мир, 1991. Пер. с англ. ISBN: 5-03-001619-8.
- [31] **Wolf-Gladrow, Dieter.** Lattice gas cellular automata and lattice Boltzmann models: an introduction. Berlin: Springer-Verlag, 2000. ISBN: 3-540-66973-6.
- [32] **Hyötyniemi, Heikki.** Complex systems: science on the edge of chaos. Helsinki: Helsinki University of Technology, Control Engineering Laboratory, Report 145, 2004, pp. 73-87,
- [33] **von Neumann, John.** The general and logical theory of automata. [book auth.] Taub, A.H. John von Neumann Collected Works. New York: Pergamon Press, Vols. Design of Computers, Theory of Automata and Numerical Analysis, 1963, pp. 288-329.
- [34] **von Neumann, John.** The theory of self-reproducing automata. [ed.] Burks A.W. Urbana: University of Illinois Press, 1966.
- [35] **Zuse, Konrad.** Rechnender Raum. Braunschweig: Vieweg & Sohn, 1969. ISBN 3-528-09609-8.
- [36] Translation, MIT Technical. Calculating Space - Konrad Zuse. Cambridge : Massachusetts Institute of Technology , 1970.
- [37] **Sarkar, Palash.** A brief history of cellular automata. ACM Computing Surveys. Vol. 32, Issue 1, 2000, pp. 80-107.
- [38] **Gardner, Martin.** Mathematical games: The fantastic combinations of John Conway's new solitaire game "Life". Scientific American. Vol. 223, Issue 4, 1970, pp. 120-123.
- [39] **Lindenmayer, Aristid.** Mathematical models for cellular interaction in development I. Filaments with one-sided inputs. Journal of Theoretical Biology. Vol. 18, Issue 3, 1968, pp. 280-315.
- [40] **Rosen, Robert.** Pattern Generation in Networks. Progress in Theoretical Biology. Vol. 6, 1981, pp. 497-525.
- [41] **Stanley, H. Eugene.** Introduction to Phase Transitions and Critical Phenomena. Oxford: Oxford University Press, 1971. ISBN: 01995014588.
- [42] **Kadanoff, P. Leo and Swift, Jack.** Transport coefficients near the critical point: a master-equation approach. Physical Review. Vol. 165, Issue 1, 1968, pp. 310-322.
- [43] **Rothman, Daniel H. and Zaleski, Stéphane.** Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flows. Reviews of Modern Physics. Vol. 66, Issue 4, 1994, pp. 1417-1479.
- [44] **Hardy, J., Pomeau, Y. and de Pazzis, O.** Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions. Journal of Mathematical Physics. Vol. 14, Issue 12, 1973, pp. 1746-1759.
- [45] **Lawson, Mark V.** Finite automata. Chapman & Hall / CRC Press, 2003. ISBN: 1-58488-255-7.

- [46] **Rivet, Jean-Pierre and Boon, Jean Pierre.** Lattice Gas Hydrodynamics. Cambridge: Cambridge University Press, 2001. ISBN: 0-521-41944-1.
- [47] **Кудрявцев, В.Б., Алёшин, С.В. и Подколзин, А.С.** Введение в теорию автоматов. Москва: Наука, 1985.
- [48] **Chytil, M.** Automaty a gramatiky. Praha: SNTL - Nakladatelství technické literatury, 1984. pp. 15-21.
- [49] **Chopard, Bastien and Droz, Michel.** Cellular Automata Modeling of Physical Systems. Cambridge: Cambridge University Press, 2005. ISBN 13-978-0-521-67345-7.
- [50] **Frisch, U, Hasslacher, B and Pomeau, Y.** Lattice-gas automata for Navier-Stokes Equation. Physical Review Letters. Vol. 56, Issue 14, 1986, pp. 1505-1508.
- [51] **Chen, Shiyi, Doolen, Cary D. and Eggert, Kenneth G.** Lattice-Boltzmann fluid dynamics. A versatile tool for multiphase and other complicated flows. Los Alamos Science. Vol. 22, 1994, pp. 99-111.
- [52] **Boublík, Tomáš.** Statistická termodynamika. Praha : Academia, 1996. ISBN 80-200-0566-8.
- [53] Discretization. Wikipedia – the free encyclopedia. [Online] [Cited: 28.01.2008] <http://en.wikipedia.org/wiki/Discretization>.
- [54] Diskretizace. COTOJE. [Online] [Cited: 01.02.2008] Zdroj: Malá Československá encyklopedie. [www.cotoje.cz](http://www.cotoje.cz).
- [55] **Cetin, Nurhan.** Discretization methods. Ph.D thesis, ETH Zurich, 2000. [Online] [Cited: 28.01.2008] <http://www.inf.ethz.ch>.
- [56] **Cetin, Nurhan.** Mesh generation. ETH Zurich. [Online] [Cited: 28.01.2008] <http://www.inf.ethz.ch/personal/cetin/thesis/thesis/node18.html>.
- [57] **Edelman, Alan.** Lecture notes in Applied Parallel Computing. MITOPENCOURSEWARE Massachusetts Institute of Technology. [Online] 2004. [Cited: 29.1.2008] Chapter 11: Mesh generation. <http://ocw.mit.edu/OcwWeb/Mathematics/18-337JSpring-2005/LectureNotes/>.
- [58] **Cho, W. and Patrikalakis, N.M.** Computational geometry. Lecture 23. [Online] 2003. [Cited: 29.1.2008] [www.ocw.mit.edu](http://www.ocw.mit.edu).
- [59] **Wang, Z. J., et al.** An enriched hybrid grid approach for unsteady multi-body flow computation. AIAA Applied Aerodynamics Conference, 12th. 1994.
- [60] Lattice (group). Wikipedia - the free encyclopedia. [Online] [Cited: 16.02.2008] <http://en.wikipedia.org>.
- [61] Lattice. The Free Dictionary. [Online] [Cited: 18.02.2008] [www.thefreedictionary.com](http://www.thefreedictionary.com).
- [62] Bravais lattice. Wikipedia - the free encyclopedia. [Online] [Cited: 18.02.2008] <http://en.wikipedia.org>.

- [63] **N. Mermin, David.** Copernican Crystallography. Physical Review Letters. 1992, Vol. 68, 8, pp. 1172-1175.
- [64] Square lattice. Wikipedia - the free encyclopedia. [Online] [Cited: 01.05.2009] <http://en.wikipedia.org>.
- [65] **Ali, S. Mustafa.** Games of Proto-Life in Masked Cellular Automata (MCA). [book auth.] R.J. Stonier and X.H. Yu. Complex Systems - Mechanism of Adaptation: IOS Press, 1994, pp. 77-84.
- [66] **Mersereau, Russell M.** The processing of hexagonally sampled two-dimensional signals. Proceedings of the IEEE. 67, 1979, pp. 930-949.
- [67] **Staunton, Richard C. a Storey, Neil.** A comparison between square and hexagonal sampling methods for pipeline image processing. Proc. SPIE. 1194, 1989, pp. 142-151.
- [68] **Weimar, Jörg R.** Simulation with Cellular Automata. Berlin: Logos-Verlag, 1997. ISBN 3-89722-026-1.
- [69] **Wolfram, Stephen.** Cellular automata fluids 1: Basic theory. Journal of Statistical Physics. Vol. 45, 3/4, 1986, pp. 471-526.
- [70] **Kroc, Jiri.** Effect of lattice anisotropy on simulations of grain boundary movement in two-dimensions. [ed.] Bacroix, B. and other. Materials Science Forum. Recrystallization and Grain Growth, Vols. 467-470, 2004, pp. 1069-1074.
- [71] **Chen, S., Doolen, G.D. and Eggert, K.G.** Lattice-Boltzmann fluid dynamics. Los Alamos Science. 22, 1994, pp. 100-109.
- [72] **Nenadál, Karel; Václavíková, Dana.** Turbo C: popis jazyka. Praha: Grada, 1991. ISBN 80-85424-08-8.
- [73] cplusplus.com. C Library. [Online] 2012. [Citate: 08.05.2012] <http://www.cplusplus.com/reference/clibrary/>.
- [74] **Wolfram, Stephen.** A New Kind of Science: Wolfram Media, Inc., 2002. ISBN 1-57955-008-8.
- [75] **Roache, Patrick J.** Verification and Validation in Computational Science and Engineering: Hermosa Publishers, 1998. ISBN-10: 0913478083.
- [76] Brownian motion. Wikipedia. The Free Encyclopedia. [Online] 2012. March 1. [Cited: 10.08.2012] [http://en.wikipedia.org/wiki/Brownian\\_motion](http://en.wikipedia.org/wiki/Brownian_motion).
- [77] **Xing, Keqiang.** Numerical Investigation on the Heat Transfer Enhancement Using Micro/Nano Phase-Change Particular Flow. FIU Electronic theses and dissertations. [Online] 2007. [Cited: 11.03.2012.] <http://digitalcommons.fiu.edu/etd/28/>.
- [78] **Bespalko, Dustin John.** Validation of the Lattice Boltzmann Method for Direct Numerical Simulation of Wall-Bounded Turbulent Flows. PhD thesis. Kingston, Ontario, Canada: Queen's University, 2011.

- [79] **Rothmann, Daniel H.** Cellular automata fluids: a model for fluid flow in porous media. *Geophysics*. 53, 1988, pp. 509-518.
- [80] **Chen, Shiyi, Doolen, G. D. and W.H., Matthaeus.** Lattice gas automata for simple and complex fluids. *Journal of Statistical Physics*. 64, 1991, Vol. No. 5/6, pp. 1133-1162.
- [81] **Yang, Z.L., et al.** Evaluation of the Darcy's Law performance for two-fluid flow hydrodynamics in a particle debris bed using lattice-Boltzmann model. *Heat and Mass Transfer*. Vol. 36, 2000, pp. 295-304.
- [82] **Kadanoff, L.P., McNamara, G.R. and Zanetti, G.** From automata to fluid flow: comparison of simulation and theory. *Physical Review A*. Vol. 40, 1989, pp. 4527-4541.
- [83] **McNamara, G. and G., Zanetti.** Direct measure of viscosity in a lattice gas model . MIT Lab for Comp: Cellular Automata '86 (abstract), 1986.
- [84] **Hrůza, Jakub.** Zlepšování filtračních vlastností vlákných materiálů. Liberec: Fakulta textilní, Technická Universita v Liberci, 2005. Disertační práce.
- [85] **Brown, R.C.** Air filtration. An Integrated Approach to the Theory and Applications of Fibrous Filters: Pergamon Press, 1993, pp. 62-64. ISBN 0 08 041274 2.
- [86] Unit of time (second). BIPM metrology portal. [Online] [Cited: 11.01.2009] [http://www.bipm.org/en/si/si\\_brochure/chapter2/2-1/second.html](http://www.bipm.org/en/si/si_brochure/chapter2/2-1/second.html).
- [87] Regular grid. Wikipedia - the free encyclopedia. [Online] [Cited: 16.02.2008] <http://en.wikipedia.org>.
- [88] Unstructured grid. Wikipedia - the free encyclopedia. [Online] [Cited: 16.02.2008] <http://en.wikipedia.org>.
- [89] **Meyer, Peter.** Lattice Geometries. Hermetic Systems. [Online] 17 02 2001. [Cited: 11.11. 2008] <http://www.hermetic.ch/compsci/lattgeom.htm>.
- [90] **Hermann, Andreas, Lein, Matthias and Schwerdtfeger, Peter.** The Search for the Species with the Highest Coordination Number. *Angewandte Chemie*. Vol. 46, 2007, pp. 2444 –2447.
- [91] Basis. Wikipedia- the free encyclopedia. [Online] [Cited: 12.02.2008] <http://en.wikipedia.org>.
- [92] **Ocheretna, L., Lukas, D.** Fluid modelling: from molecular level to continuum behaviour in porous materials. 8 pages, Proceedings edited by Riitta Salonen & Pirjo Heikkilä, Autex 2007, Tampere, Finland, 26-28 June, ISBN 978-952-15-1794-5.

## PUBLICATIONS OF AUTHOR

1. **Ocheretna, L.** Computer simulation of fluid flow through porous media. Strutex 03, Liberec – Česká republika, 2003, pp. 65-69, ISBN 80-7083-769-1.
2. **Očeretná, L.** Teorie buněčných automatů. Klas modelu FHP. Písemná práce ke zkoušce „Vybrané partie z teorie oboru“, FT TUL, Liberec 2003.
3. **Ocheretna, L.** Modelling of textile materials' physical properties: usage of cellular automata method. Strutex 04, Liberec – Česká republika, 2004, pp. 159-163, ISBN 80-7083-891-4.
4. **Lukáš, D., Košťáková, E., Chaloupek, J., Očeretna, L., Pociute, M.** Instability of Liquid Jets. Strutex 04, Liberec – Česká republika, 2004.
5. **Očeretná, L.** Testování generátoru pseudonáhodných čísel použitého v simulačním modelu FHP. Písemná práce ke zkoušce „Přírodovědecký základ. Základy matematické statistiky“, FT TUL, Liberec 2004.
6. **Ocheretna, L.** Modeling of generation and propagation of harmonic waves based on a FHP lattice gas model. MOSIS'05, Hradec nad Moravicí, Česká republika, 2005, pp. 313-318.
7. **Ocheretna, L., Lukáš, D.** Modeling of ultrasound wave motion by means of FHP lattice gas model. AUTEX'05, Portorož, Slovinsko, 2005, pp. 634-639.
8. **Košťáková, E., Grégr, J., Očeretna, L.** Nanovlákná a možnosti jejich uplatnění v kompozitních materiálech. Vyztužené plasty 2005, Karlovy Vary, Česká republika 2005.
9. **Ocheretna, L., Košťáková, E.** Ultrasound and Textile Technology – Cellular Automata Simulation and Experiments. Proceedings of ForumAcusticum, Budapest, Hungary, 29 Aug-2 Sep, 2005, pp. 2843-2848 .
10. **Ocheretna, L.** Using of lattice gas cellular automata for textile material's physical properties modelling. International Summer Conference-School “Advanced Materials and Technologies”, Palanga, Lithuania, 27-31 August 2006, ISBN 9955-25-101-8.
11. **Lukas, D., and Ocheretna, L.** The cellular automata lattice gas approach for fluid flows in porous media. [ed.] N. Pan and P. Gibson. Thermal and moisture transport in fibrous materials. Cambridge: Woohnead Publishing Limited, 2006, pp. 357-401.
12. **Ocheretna, L., Lukas, D.** Fluid modelling: from molecular level to continuum behaviour in porous materials. 8 pages, Proceedings edited by Riitta Salonen & Pirjo Heikkilä, Autex 2007, Tampere, Finland, 26-28 June, ISBN 978-952-15-1794-5.
13. **Ocheretna, L., Lukáš, D.** Modelling of diffusivity by means of 2-D lattice gas cellular automata model. Book of abstracts, 6th international conference Textile Science (TEXSCI) 2007, Liberec, Czech Republic, 5-7 June, ISBN 978-80-7372-207-4.
14. **Ocheretna, L.** Diffusivity and diffusion coefficient in two-dimensional lattice gas cellular automata. The 9-th International Conference-School “Advanced materials and technologies”, Palanga, Lithuania, 27-31 August 2007, ISSN 1822-7759.
15. **Ocheretna, L.** Lattice gas cellular automata as an alternative for fluid flow modelling. Písemná práce k SDZ, FT TUL, Liberec 2009.

16. **Lukas ,D., Pan, N., Sarkar, A., Weng, M., Chaloupek, J., Kostakova, E., Ocheretna, L., Mikes, P., Pociute, M., Amler, E.** Auto-model based computer simulation of Plateau–Rayleigh instability of mixtures of immiscible liquids. *Physica A: Statistical Mechanics and its Applications*, Volume 389, Issue 11, 1 June 2010, pp. 2164-2176.

## LIST OF APPENDIXES

<b>Appendix A:</b> The FHP-1 Lattice Cellular Automata model. The list of possible pre- and post-collision states of an individual automaton .....	123
<b>Appendix B:</b> The FHP-2 Lattice Cellular Automata model. The list of possible pre- and post-collision states of an individual automaton.....	129
<b>Appendix C:</b> Basic FHP-1 Lattice Gas Cellular Automata algorithm.....	139
<b>Appendix D:</b> Time evolution of the FHP-1 Lattice Gas Cellular Automata model.....	148
<b>Appendix E:</b> The FHP-1 Lattice Gas Cellular Automata algorithm for Brownian motion simulation .....	152
<b>Appendix F:</b> Computer simulation of the Brownian motion. Evolution in time for 20 time steps .....	165
<b>Appendix G:</b> Computer simulation of the Brownian motion. Paths of the Brownian particle after 4000 time steps. Experiment 1 .....	170
<b>Appendix H:</b> Computer simulation of the Brownian motion. Paths of the Brownian particle after 4000 time steps. Experiment 2 .....	173
<b>Appendix I:</b> The FHP-1 Lattice Gas Cellular Automata for Poiseuille flow simulation .....	176
<b>Appendix J:</b> Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Flow rate as a function of time and velocity profiles for various width of the channel $d$ .....	190
<b>Appendix K:</b> Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Parabolic velocity profiles of the flow for various width of the channel $d$ and for various values of the parameter $f_x$ .....	195
<b>Appendix L:</b> Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Flow rate as a function of channel width $d$ for pressure gradient created by various $f_x$ .....	199
<b>Appendix M:</b> Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Validation of the Darcy's law for various width of the channel $d$ .....	201
<b>Appendix N:</b> The FHP-1 Lattice Gas Cellular Automata algorithm for simulation of the fluid flow through porous media .....	203
<b>Appendix O:</b> The FHP-1 Lattice Gas Cellular Automata algorithm for simulation of the fluid flow through porous media. Algorithms for data processing and their graphical representation.....	219
<b>Appendix P:</b> Computer simulation of the fluid flow through declined porous media. Evolution in time for a period of 20 time steps.....	224+
<b>Appendix Q:</b> Computer simulation of the fluid flow through declined porous media. Flow rate as a function of time for various inclination angle $\alpha$ .....	228
<b>Appendix R:</b> Computer simulation of the fluid flow through declined porous media. Time evolution of the system with reduced simulation domain .....	229
<b>Appendix S:</b> Computer simulation of the fluid flow through declined porous media. Fields of velocity vectors. ....	233

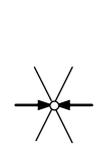
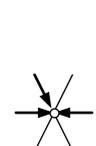
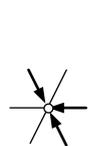
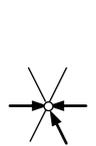
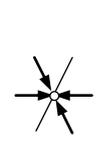
## APPENDIX A

### The FHP-1 Lattice Gas Cellular Automata model The list of possible pre- and post-collision states of an individual automaton

---

In the table  $i_1 \dots i_6$  corresponds to six channels of the particular individual automaton (lattice node). Values "0" and "1" are in accordance with the channel occupation. When the channel is occupied it gets value 1, other way it remains "0".

Graphical interpretation of pre-collision states in a node	Representation of the pre-collision state							Graphical interpretation of post-collision states in a node	Representation of the post-collision state							Efficiency of the collision
	Binary representation of the state						Coding of the state using decimal system		Binary representation of the state						Coding of the state using decimal system	
	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$			$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$		
	0	0	0	0	0	0	0		0	0	0	0	0	0	0	
	1	0	0	0	0	0	1		0	0	0	1	0	0	8	
	0	1	0	0	0	0	2		0	0	0	0	1	0	16	
	1	1	0	0	0	0	3		0	0	0	1	1	0	24	
	0	0	1	0	0	0	4		0	0	0	0	0	1	32	
	1	0	1	0	0	0	5		0	0	0	1	0	1	40	
	0	1	1	0	0	0	6		0	0	0	0	1	1	48	
	1	1	1	0	0	0	7		0	0	0	1	1	1	56	
	0	0	0	1	0	0	8		1	0	0	0	0	0	1	
	1	0	0	1	0	0	9		0	1	0	0	1	0	18	Efficient collision
									0	0	1	0	0	1	36	
	0	1	0	1	0	0	10		1	0	0	0	1	0	17	
	1	1	0	1	0	0	11		1	0	0	1	1	0	25	
									0	0	1	0	1	1	52	
	0	0	1	1	0	0	12		1	0	0	0	0	1	33	
	1	0	1	1	0	0	13		1	0	0	1	0	1	41	
									0	1	0	0	1	1	50	

	0	1	1	1	0	0	<b>14</b>		1	0	0	0	1	1	<b>49</b>	
	1	1	1	1	0	0	<b>15</b>		1	0	0	1	1	1	<b>57</b>	
	0	0	0	0	1	0	<b>16</b>		0	1	0	0	0	0	<b>2</b>	
	1	0	0	0	1	0	<b>17</b>		0	1	0	1	0	0	<b>10</b>	
	0	1	0	0	1	0	<b>18</b>		0	0	1	0	0	1	<b>36</b>	Efficient collision
									1	0	0	1	0	0	<b>9</b>	
	1	1	0	0	1	0	<b>19</b>		0	1	0	1	1	0	<b>26</b>	
									0	0	1	1	0	1	<b>44</b>	
	0	0	1	0	1	0	<b>20</b>		0	1	0	0	0	1	<b>34</b>	
	1	0	1	0	1	0	<b>21</b>		0	1	0	1	0	1	<b>42</b>	Efficient collision
	0	1	1	0	1	0	<b>22</b>		0	1	0	0	1	1	<b>50</b>	
									1	0	0	1	0	1	<b>41</b>	
	1	1	1	0	1	0	<b>23</b>		0	1	0	1	1	1	<b>58</b>	
	0	0	0	1	1	0	<b>24</b>		1	1	0	0	0	0	<b>3</b>	
	1	0	0	1	1	0	<b>25</b>		1	1	0	1	0	0	<b>11</b>	
									0	1	1	0	0	1	<b>38</b>	
	0	1	0	1	1	0	<b>26</b>		1	1	0	0	1	0	<b>19</b>	
									1	0	1	0	0	1	<b>37</b>	
	1	1	0	1	1	0	<b>27</b>		1	0	1	1	0	1	<b>45</b>	
									0	1	1	0	1	1	<b>54</b>	

	0	0	1	1	1	0	<b>28</b>		1	1	0	0	0	1	<b>35</b>	
	1	0	1	1	1	0	<b>29</b>		1	1	0	1	0	1	<b>43</b>	
	0	1	1	1	1	0	<b>30</b>		1	1	0	0	1	1	<b>51</b>	
	1	1	1	1	1	0	<b>31</b>		1	1	0	1	1	1	<b>59</b>	
	0	0	0	0	0	1	<b>32</b>		0	0	1	0	0	0	<b>4</b>	
	1	0	0	0	0	1	<b>33</b>		0	0	1	1	0	0	<b>12</b>	
	0	1	0	0	0	1	<b>34</b>		0	0	1	0	1	0	<b>20</b>	
	1	1	0	0	0	1	<b>35</b>		0	0	1	1	1	0	<b>28</b>	
	0	0	1	0	0	1	<b>36</b>		1	0	0	1	0	0	<b>9</b>	Efficient collision
									0	1	0	0	1	0	<b>18</b>	
	1	0	1	0	0	1	<b>37</b>		0	0	1	1	0	1	<b>44</b>	
									0	1	0	1	1	0	<b>26</b>	
	0	1	1	0	0	1	<b>38</b>		0	0	1	0	1	1	<b>52</b>	
									1	0	0	1	1	0	<b>25</b>	
	1	1	1	0	0	1	<b>39</b>		0	0	1	1	1	1	<b>60</b>	
	0	0	0	1	0	1	<b>40</b>		1	0	1	0	0	0	<b>5</b>	
	1	0	0	1	0	1	<b>41</b>		1	0	1	1	0	0	<b>13</b>	
									0	1	1	0	1	0	<b>22</b>	
	0	1	0	1	0	1	<b>42</b>		1	0	1	0	1	0	<b>21</b>	Efficient collision
	1	1	0	1	0	1	<b>43</b>		1	0	1	1	1	0	<b>29</b>	

	0	0	1	1	0	1	<b>44</b>		1	0	1	0	0	1	<b>37</b>	
										1	1	0	0	1	0	
	1	0	1	1	0	1	<b>45</b>		1	1	0	1	1	0	<b>27</b>	
										0	1	1	0	1	1	
	0	1	1	1	0	1	<b>46</b>		1	0	1	0	1	1	<b>53</b>	
	1	1	1	1	0	1	<b>47</b>		1	0	1	1	1	1	<b>61</b>	
	0	0	0	0	1	1	<b>48</b>		0	1	1	0	0	0	<b>6</b>	
	1	0	0	0	1	1	<b>49</b>		0	1	1	1	0	0	<b>14</b>	
	0	1	0	0	1	1	<b>50</b>		0	1	1	0	1	0	<b>22</b>	
										1	0	1	1	0	0	
	1	1	0	0	1	1	<b>51</b>		0	1	1	1	1	0	<b>30</b>	
	0	0	1	0	1	1	<b>52</b>		0	1	1	0	0	1	<b>38</b>	
										1	1	0	1	0	0	
	1	0	1	0	1	1	<b>53</b>		0	1	1	1	0	1	<b>46</b>	
	0	1	1	0	1	1	<b>54</b>		1	0	1	1	0	1	<b>45</b>	
										1	1	0	1	1	0	
	1	1	1	0	1	1	<b>55</b>		0	1	1	1	1	1	<b>62</b>	
	0	0	0	1	1	1	<b>56</b>		1	1	1	0	0	0	<b>7</b>	
	1	0	0	1	1	1	<b>57</b>		1	1	1	1	0	0	<b>15</b>	
	0	1	0	1	1	1	<b>58</b>		1	1	1	0	1	0	<b>23</b>	

	1	1	0	1	1	1	<b>59</b>		1	1	1	1	1	0	<b>31</b>	
	0	0	1	1	1	1	<b>60</b>		1	1	1	0	0	1	<b>39</b>	
	1	0	1	1	1	1	<b>61</b>		1	1	1	1	0	1	<b>47</b>	
	0	1	1	1	1	1	<b>62</b>		1	1	1	0	1	1	<b>55</b>	
	1	1	1	1	1	1	<b>63</b>		1	1	1	1	1	1	<b>63</b>	

**APPENDIX B**

## The FHP-2 Lattice Gas Cellular Automata model. The list of possible pre- and post-collision states of an individual automaton

---

In the table  $i_1 \dots i_7$  corresponds to seven channels of the particular individual automaton (lattice node). Values "0" and "1" are in accordance with the channel occupation. When the channel is occupied it gets value 1, other way it remains "0".

Graphical interpretation of pre-collision states in a node	Representation of the pre-collision state								Graphical interpretation of post-collision states in a node	Representation of the post-collision state								Efficiency of the collision	
	Binary representation of the state							Coding of the state using decimal system		Binary representation of the state							Coding of the state using decimal system		
	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$			$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$			
	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0		
	1	0	0	0	0	0	0	1		0	0	0	1	0	0	0	0	8	
	0	1	0	0	0	0	0	2		0	0	0	0	1	0	0	0	16	
	1	1	0	0	0	0	0	3		0	0	0	1	1	0	0	0	24	
	0	0	1	0	0	0	0	4		0	0	0	0	0	1	0	0	32	
	1	0	1	0	0	0	0	5		0	0	0	0	1	0	1	0	80	Efficient collision
	0	1	1	0	0	0	0	6		0	0	0	0	1	1	0	0	48	
	1	1	1	0	0	0	0	7		0	0	0	1	1	1	0	0	56	
	0	0	0	1	0	0	0	8		1	0	0	0	0	0	0	0	1	
	1	0	0	1	0	0	0	9		0	1	0	0	1	0	0	0	18	Efficient collision
										0	0	1	0	0	1	0	0	36	
	0	1	0	1	0	0	0	10		0	0	0	0	0	1	1	0	96	Efficient collision
	1	1	0	1	0	0	0	11		1	0	0	1	1	0	0	0	25	
										0	0	1	0	1	1	0	0	52	
	0	0	1	1	0	0	0	12		1	0	0	0	0	1	0	0	33	
	1	0	1	1	0	0	0	13		1	0	0	1	0	1	0	0	41	
										0	1	0	0	1	1	0	0	50	
	0	1	1	1	0	0	0	14		1	0	0	0	1	1	0	0	49	

	1	1	1	1	0	0	0	15		1	0	0	1	1	1	0	57	
	0	0	0	0	1	0	0	16		0	1	0	0	0	0	0	2	
	1	0	0	0	1	0	0	17		0	0	1	0	0	0	1	68	Efficient collision
	0	1	0	0	1	0	0	18		0	0	1	0	0	1	0	36	Efficient collision
										1	0	0	1	0	0	0	9	
	1	1	0	0	1	0	0	19		0	1	0	1	1	0	0	26	
										0	0	1	1	0	1	0	44	
	0	0	1	0	1	0	0	20		1	0	0	0	0	0	1	65	Efficient collision
	1	0	1	0	1	0	0	21		0	1	0	1	0	1	0	42	Efficient collision
	0	1	1	0	1	0	0	22		0	1	0	0	1	1	0	50	
										1	0	0	1	0	1	0	41	
	1	1	1	0	1	0	0	23		0	1	0	1	1	1	0	58	
	0	0	0	1	1	0	0	24		1	1	0	0	0	0	0	3	
	1	0	0	1	1	0	0	25		1	1	0	1	0	0	0	11	
										0	1	1	0	0	1	0	38	
	0	1	0	1	1	0	0	26		1	1	0	0	1	0	0	19	
										1	0	1	0	0	1	0	37	
	1	1	0	1	1	0	0	27		1	0	1	1	0	1	0	45	
										0	1	1	0	1	1	0	54	
	0	0	1	1	1	0	0	28		1	1	0	0	0	1	0	35	

	1	0	1	1	1	0	0	<b>29</b>		1	1	0	1	0	1	0	<b>43</b>	
	0	1	1	1	1	0	0	<b>30</b>		1	1	0	0	1	1	0	<b>51</b>	
	1	1	1	1	1	0	0	<b>31</b>		1	1	0	1	1	1	0	<b>59</b>	
	0	0	0	0	0	1	0	<b>32</b>		0	0	1	0	0	0	0	<b>4</b>	
	1	0	0	0	0	1	0	<b>33</b>		0	0	1	1	0	0	0	<b>12</b>	
	0	1	0	0	0	1	0	<b>34</b>		0	0	0	1	0	0	1	<b>72</b>	Efficient collision
	1	1	0	0	0	1	0	<b>35</b>		0	0	1	1	1	0	0	<b>28</b>	
	0	0	1	0	0	1	0	<b>36</b>		1	0	0	1	0	0	0	<b>9</b>	Efficient collision
										0	1	0	0	1	0	0	<b>18</b>	
	1	0	1	0	0	1	0	<b>37</b>		0	0	1	1	0	1	0	<b>44</b>	
										0	1	0	1	1	0	0	<b>26</b>	
	0	1	1	0	0	1	0	<b>38</b>		0	0	1	0	1	1	0	<b>52</b>	
										1	0	0	1	1	0	0	<b>25</b>	
	1	1	1	0	0	1	0	<b>39</b>		0	0	1	1	1	1	0	<b>60</b>	
	0	0	0	1	0	1	0	<b>40</b>		0	1	0	0	0	0	1	<b>66</b>	Efficient collision
	1	0	0	1	0	1	0	<b>41</b>		1	0	1	1	0	0	0	<b>13</b>	
										0	1	1	0	1	0	0	<b>22</b>	
	0	1	0	1	0	1	0	<b>42</b>		1	0	1	0	1	0	0	<b>21</b>	Efficient collision
	1	1	0	1	0	1	0	<b>43</b>		1	0	1	1	1	0	0	<b>29</b>	
	0	0	1	1	0	1	0	<b>44</b>		1	0	1	0	0	1	0	<b>37</b>	

										1	1	0	0	1	0	0	<b>19</b>	
	1	0	1	1	0	1	0	<b>45</b>		1	1	0	1	1	0	0	<b>27</b>	
										0	1	1	0	1	1	0	<b>54</b>	
	0	1	1	1	0	1	0	<b>46</b>		1	0	1	0	1	1	0	<b>53</b>	
	1	1	1	1	0	1	0	<b>47</b>		1	0	1	1	1	1	0	<b>61</b>	
	0	0	0	0	1	1	0	<b>48</b>		0	1	1	0	0	0	0	<b>6</b>	
	1	0	0	0	1	1	0	<b>49</b>		0	1	1	1	0	0	0	<b>14</b>	
	0	1	0	0	1	1	0	<b>50</b>		0	1	1	0	1	0	0	<b>22</b>	
										1	0	1	1	0	0	0	<b>13</b>	
	1	1	0	0	1	1	0	<b>51</b>		0	1	1	1	1	0	0	<b>30</b>	
	0	0	1	0	1	1	0	<b>52</b>		0	1	1	0	0	1	0	<b>38</b>	
										1	1	0	1	0	0	0	<b>11</b>	
	1	0	1	0	1	1	0	<b>53</b>		0	1	1	1	0	1	0	<b>46</b>	
	0	1	1	0	1	1	0	<b>54</b>		1	0	1	1	0	1	0	<b>45</b>	
										1	1	0	1	1	0	0	<b>27</b>	
	1	1	1	0	1	1	0	<b>55</b>		0	1	1	1	1	1	0	<b>62</b>	
	0	0	0	1	1	1	0	<b>56</b>		1	1	1	0	0	0	0	<b>7</b>	
	1	0	0	1	1	1	0	<b>57</b>		1	1	1	1	0	0	0	<b>15</b>	
	0	1	0	1	1	1	0	<b>58</b>		1	1	1	0	1	0	0	<b>23</b>	
	1	1	0	1	1	1	0	<b>59</b>		1	1	1	1	1	0	0	<b>31</b>	

	0	0	1	1	1	1	0	<b>60</b>		1	1	1	0	0	1	0	<b>39</b>	
	1	0	1	1	1	1	0	<b>61</b>		1	1	1	1	0	1	0	<b>47</b>	
	0	1	1	1	1	1	0	<b>62</b>		1	1	1	0	1	1	0	<b>55</b>	
	1	1	1	1	1	1	0	<b>63</b>		1	1	1	1	1	1	0	<b>63</b>	
	0	0	0	0	0	0	1	<b>64</b>		0	0	0	0	0	0	1	<b>64</b>	
	1	0	0	0	0	0	1	<b>65</b>		0	0	1	0	1	0	0	<b>20</b>	Efficient collision
	0	1	0	0	0	0	1	<b>66</b>		0	0	0	1	0	1	0	<b>40</b>	Efficient collision
	1	1	0	0	0	0	1	<b>67</b>		0	0	0	1	1	0	1	<b>88</b>	
	0	0	1	0	0	0	1	<b>68</b>		1	0	0	0	1	0	0	<b>17</b>	Efficient collision
	1	0	1	0	0	0	1	<b>69</b>		0	0	0	1	0	1	1	<b>104</b>	
	0	1	1	0	0	0	1	<b>70</b>		0	0	0	0	1	1	1	<b>112</b>	
	1	1	1	0	0	0	1	<b>71</b>		0	0	0	1	1	1	1	<b>120</b>	
	0	0	0	1	0	0	1	<b>72</b>		0	1	0	0	0	1	0	<b>34</b>	Efficient collision
	1	0	0	1	0	0	1	<b>73</b>		0	1	0	0	1	0	1	<b>82</b>	Efficient collision
										0	0	1	0	0	1	1	<b>100</b>	
	0	1	0	1	0	0	1	<b>74</b>		1	0	0	0	1	0	1	<b>81</b>	
	1	1	0	1	0	0	1	<b>75</b>		1	0	0	1	1	0	1	<b>89</b>	
										0	0	1	0	1	1	1	<b>116</b>	
	0	0	1	1	0	0	1	<b>76</b>		1	0	0	0	0	1	1	<b>97</b>	
	1	0	1	1	0	0	1	<b>77</b>		1	0	0	1	0	1	1	<b>105</b>	

										0	1	0	0	1	1	1	<b>114</b>	
	0	1	1	1	0	0	1	<b>78</b>		1	0	0	0	1	1	1	<b>113</b>	
	1	1	1	1	0	0	1	<b>79</b>		1	0	0	1	1	1	1	<b>121</b>	
	0	0	0	0	1	0	1	<b>80</b>		1	0	1	0	0	0	0	<b>5</b>	Efficient collision
	1	0	0	0	1	0	1	<b>81</b>		0	1	0	1	0	0	1	<b>74</b>	
	0	1	0	0	1	0	1	<b>82</b>		0	0	1	0	0	1	1	<b>100</b>	Efficient collision
										1	0	0	1	0	0	1	<b>73</b>	
	1	1	0	0	1	0	1	<b>83</b>		0	1	0	1	1	0	1	<b>90</b>	
										0	0	1	1	0	1	1	<b>108</b>	
	0	0	1	0	1	0	1	<b>84</b>		0	1	0	0	0	1	1	<b>98</b>	
	1	0	1	0	1	0	1	<b>85</b>		0	1	0	1	0	1	1	<b>106</b>	Efficient collision
										1	0	1	0	1	0	1	<b>85</b>	
	0	1	1	0	1	0	1	<b>86</b>		0	1	0	0	1	1	1	<b>114</b>	
										1	0	0	1	0	1	1	<b>105</b>	
	1	1	1	0	1	0	1	<b>87</b>		0	1	0	1	1	1	1	<b>122</b>	
	0	0	0	1	1	0	1	<b>88</b>		1	1	0	0	0	0	1	<b>67</b>	
	1	0	0	1	1	0	1	<b>89</b>		1	1	0	1	0	0	1	<b>75</b>	
										0	1	1	0	0	1	1	<b>102</b>	
	0	1	0	1	1	0	1	<b>90</b>		1	1	0	0	1	0	1	<b>83</b>	
										1	0	1	0	0	1	1	<b>101</b>	

	1	1	0	1	1	0	1	<b>91</b>		1	0	1	1	0	1	1	<b>109</b>	
											0	1	1	0	1	1	1	
	0	0	1	1	1	0	1	<b>92</b>		1	1	0	0	0	1	1	<b>99</b>	
	1	0	1	1	1	0	1	<b>93</b>		1	1	0	1	0	1	1	<b>107</b>	
	0	1	1	1	1	0	1	<b>94</b>		1	1	0	0	1	1	1	<b>115</b>	
	1	1	1	1	1	0	1	<b>95</b>		1	1	0	1	1	1	1	<b>123</b>	
	0	0	0	0	0	1	1	<b>96</b>		0	1	0	1	0	0	0	<b>10</b>	Efficient collision
	1	0	0	0	0	1	1	<b>97</b>		0	0	1	1	0	0	1	<b>76</b>	
	0	1	0	0	0	1	1	<b>98</b>		0	0	1	0	1	0	1	<b>84</b>	
	1	1	0	0	0	1	1	<b>99</b>		0	0	1	1	1	0	1	<b>92</b>	
	0	0	1	0	0	1	1	<b>100</b>		1	0	0	1	0	0	1	<b>73</b>	Efficient collision
										0	1	0	0	1	0	1	<b>82</b>	
	1	0	1	0	0	1	1	<b>101</b>		0	0	1	1	0	1	1	<b>108</b>	
										0	1	0	1	1	0	1	<b>90</b>	
	0	1	1	0	0	1	1	<b>102</b>		0	0	1	0	1	1	1	<b>116</b>	
										1	0	0	1	1	0	1	<b>89</b>	
	1	1	1	0	0	1	1	<b>103</b>		0	0	1	1	1	1	1	<b>124</b>	
	0	0	0	1	0	1	1	<b>104</b>		1	0	1	0	0	0	1	<b>69</b>	
	1	0	0	1	0	1	1	<b>105</b>		1	0	1	1	0	0	1	<b>77</b>	
										0	1	1	0	1	0	1	<b>86</b>	

APPENDIX B

	0	1	0	1	0	1	1	<b>106</b>		1	0	1	0	1	0	1	<b>85</b>	Efficient collision
	1	1	0	1	0	1	1	<b>107</b>		1	0	1	1	1	0	1	<b>93</b>	
	0	0	1	1	0	1	1	<b>108</b>		1	0	1	0	0	1	1	<b>101</b>	
										1	1	0	0	1	0	1	<b>83</b>	
	1	0	1	1	0	1	1	<b>109</b>		1	1	0	1	1	0	1	<b>91</b>	
										0	1	1	0	1	1	1	<b>118</b>	
	0	1	1	1	0	1	1	<b>110</b>		1	0	1	0	1	1	1	<b>117</b>	
	1	1	1	1	0	1	1	<b>111</b>		1	0	1	1	1	1	1	<b>125</b>	
	0	0	0	0	1	1	1	<b>112</b>		0	1	1	0	0	0	1	<b>70</b>	
	1	0	0	0	1	1	1	<b>113</b>		0	1	1	1	0	0	1	<b>78</b>	
	0	1	0	0	1	1	1	<b>114</b>		0	1	1	0	1	0	1	<b>86</b>	
										1	0	1	1	0	0	1	<b>77</b>	
	1	1	0	0	1	1	1	<b>115</b>		0	1	1	1	1	0	1	<b>94</b>	
	0	0	1	0	1	1	1	<b>116</b>		0	1	1	0	0	1	1	<b>102</b>	
										1	1	0	1	0	0	1	<b>75</b>	
	1	0	1	0	1	1	1	<b>117</b>		0	1	1	1	0	1	1	<b>110</b>	
	0	1	1	0	1	1	1	<b>116</b>		1	0	1	1	0	1	1	<b>109</b>	
										1	1	0	1	1	0	1	<b>91</b>	
	1	1	1	0	1	1	1	<b>119</b>		0	1	1	1	1	1	1	<b>126</b>	
	0	0	0	1	1	1	1	<b>120</b>		1	1	1	0	0	0	1	<b>71</b>	

	1	0	0	1	1	1	1	<b>121</b>		1	1	1	1	0	0	1	<b>79</b>	
	0	1	0	1	1	1	1	<b>122</b>		1	1	1	0	1	0	1	<b>87</b>	
	1	1	0	1	1	1	1	<b>123</b>		1	1	1	1	1	0	1	<b>95</b>	
	0	0	1	1	1	1	1	<b>124</b>		1	1	1	0	0	1	1	<b>103</b>	
	1	0	1	1	1	1	1	<b>125</b>		1	1	1	1	0	1	1	<b>111</b>	
	0	1	1	1	1	1	1	<b>126</b>		1	1	1	0	1	1	1	<b>119</b>	
	1	1	1	1	1	1	1	<b>127</b>		1	1	1	1	1	1	1	<b>127</b>	

## APPENDIX C

# Basic FHP-1 Lattice Gas Cellular Automata algorithm

---

```

//Basic FHP-1 LGCA algorithm

/* Code fragment 1: Header files and initialization of a simulation box */

//Definition of standard library functions
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <float.h>
# include <time.h>
# define DIRX 300
# define DIRY 300

//Declaration of variables
int x, y, xmax=299, ymax=299;
float vx[DIRX][DIRY], nvx[DIRX][DIRY];
float vy[DIRX][DIRY], nvy[DIRX][DIRY];
int m[DIRX][DIRY], nm[DIRX][DIRY];
int i1[DIRX][DIRY], i2[DIRX][DIRY], i3[DIRX][DIRY], i4[DIRX][DIRY],
i5[DIRX][DIRY], i6[DIRX][DIRY];
float sinangle=0.866025403, print=5.5;
int pco=20;
int sig=15;
char str[25];
int I1, I2, I3, I4, I5, I6;
int cycle, cmax=20, series;

//Declaration of subroutines
int collision(void);
float propagationodd(void);
float propagationeven(void);

/*-----*/

/* Beginning of a main part of the program */
int main()
{

/* Code fragment 2: Graphic outputs setting */

int gdriver = DETECT, gmode, errorcode;

//initialize graphics and local variabls
initgraph (&gdriver, &gmode, "c:\\TC\\BGI");

//read result of initialization
errorcode = graphresult();

//an error occurred
if (errorcode != grOk)
{
printf ("Graphics error: %s\n", grapherrormsg(errorcode));
printf ("Press any key to halt:");
getch();
exit(1);
}

/* Code fragment 3: Creation of the simulation domain and initial state of
the simulated system */

```

```

//Data arrays resetting
for (x=0; x<xmax+1; x++)
    {
        for (y=0; y<ymax+1; y++)
            {
                m[x][y]=0;
                nm[x][y]=0;
                vx[x][y]=0;
                nvx[x][y]=0;
                vy[x][y]=0;
                nvy[x][y]=0;
            }
    }

//Creation of solid boundaries of the simulation box
for (x=1; x<xmax; x++)
    {
        m[x][1]=7;
        m[x][2]=7;
        m[x][ymax-1]=7;
        m[x][ymax-2]=7;

        nm[x][1]=7;
        nm[x][2]=7;
        nm[x][ymax-1]=7;
        nm[x][ymax-2]=7;
        putpixel (x, 1, m[x][1]*print);
        putpixel (x, ymax-1, m[x][ymax-1]*print);
    }

for (y=1; y<ymax; y++)
    {
        m[1][y]=7;
        m[2][y]=7;
        m[xmax-1][y]=7;
        m[xmax-2][y]=7;
        nm[1][y]=7;
        nm[2][y]=7;
        nm[xmax-1][y]=7;
        nm[xmax-2][y]=7;
        putpixel (1, y, m[1][y]*print);
        putpixel (xmax-1, y, m[xmax-1][y]*print);
    }

randomize();

/* Code fragment 4: Occupation of cannels by fluid moving particles */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=3; y<ymax-2; y=y+2)
            {
                if (m[x][y]!=7)
                    {
                        m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

                        if (m[x-1][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

                        if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}
                    }
            }
    }

```

```

        if (m[x-1][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

        if (m[x][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

        if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

        if (m[x][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

        //the total particles velocity in the node
        vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
        vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
    }}
}

//even rows of the lattice
for (x=3; x<xmax-2; x++)
{
    for (y=4; y<ymax-2; y=y+2)
    {
        if (m[x][y]!=7)
        {
            m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

            if (m[x][y-1]<7) {I1=random(pco);}
            if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else
{i1[x][y]=0;}

            if (m[x-1][y]<7) {I2=random(pco);}
            if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else
{i2[x][y]=0;}

            if (m[x][y+1]<7) {I3=random(pco);}
            if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else
{i3[x][y]=0;}

            if (m[x+1][y+1]<7) {I4=random(pco);}
            if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else
{i4[x][y]=0;}

            if (m[x+1][y]<7) {I5=random(pco);}
            if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else
{i5[x][y]=0;}

            if (m[x+1][y-1]<7) {I6=random(pco);}
            if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else
{i6[x][y]=0;}

            // the total particles velocity in the node
            vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
            vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
        }}
}

```

```
/* Code fragment 5: Graphical outputs of the initial system configuration
*/
```

```
for (x=1; x<xmax+1; x++)
{
    for (y=1; y<ymax+1; y++)
    {
        putpixel (x, y, m[x][y]*print);
    }
}
```

```
/*-----*/
```

```
/* Code fragment 6: The main cycle of the algorithm */
```

```
for (cycle=0; cycle<cmax+1; cycle++)
{
```

```
/* Code fragment 6-A: Collision phase */
```

```
for (x=3; x<xmax-2; x++)
{for (y=3; y<ymax-2; y++)
{
    if ((m[x][y]>0)&&(m[x][y]!=7)) {collision();}
}
}
```

```
/* Code fragment 6-B: Propagation phase */
```

```
//odd rows of the lattice
```

```
for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y=y+2)
    {
        if ((m[x][y]>0)&&(m[x][y]!=7))
        {
            propagationodd();
        }
    }
}
```

```
//even rows of the lattice
```

```
for (x=3; x<xmax-2; x++)
{
    for (y=4; y<ymax-2; y=y+2)
    {
        if ((m[x][y]>0)&&(m[x][y]!=7))
        {
            propagationeven();
        }
    }
}
```

```
/* Code fragment 7: Recording of a new system's state */
```

```
for (x=1; x<xmax+1; x++)
{
    for (y=1; y<ymax+1; y++)
    {
        m[x][y]=nm[x][y]; vx[x][y]=nvx[x][y]; vy[x][y]=nvy[x][y];
        putpixel (x, y, m[x][y]*print);
    }
}
```

```

/* Code fragment 8: Data arrays resetting */
for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y++)
        {
            if (nm[x][y]<7)
                {nm[x][y]=0; nvx[x][y]=0; nvy[x][y]=0;
                 i1[x][y]=0; i2[x][y]=0; i3[x][y]=0;
                 i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;}
            else {nm[x][y]=7;}
        }
}

/* Code fragment 9: Printout macro */
setfillstyle(1,0);
bar(getmaxx()-90,getmaxy()-90,getmaxx(),getmaxy()-70);
outtextxy(getmaxx()-120,getmaxy()-80,"cycle");
outtextxy(getmaxx()-70,getmaxy()-80,gcvt(series,sig,str));

series=series+1;
getch();
} //The end of the main cycle

/* Code fragment 10: Final operations */

getch();
closegraph();
return (0);
} //The end of the main part of the algorithm

/*-----*/

//SUBROUTINES

/*-----*/
//Collision phase
int collision(void)
{
int cannel=0;
int mas=0;
float velx=0;
float vely=0;

nav2:
velx=vx[x][y]; vely=vy[x][y]; mas=m[x][y];
i1[x][y]=0; i2[x][y]=0; i3[x][y]=0; i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;

nav1:
cannel=0;
cannel=random(6);

    if (cannel==0)
        {if (i1[x][y]==1) {goto nav1;}
         i1[x][y]=1; mas=mas-1;}
    if (cannel==1)
        {if (i2[x][y]==1) {goto nav1;}
         i2[x][y]=1; mas=mas-1;}
    if (cannel==2)
        {if (i3[x][y]==1) {goto nav1;}
         i3[x][y]=1; mas=mas-1;}

```

```

    if (cannel==3)
        {if (i4[x][y]==1) {goto nav1;}
        i4[x][y]=1; mas=mas-1;}
    if (cannel==4)
        {if (i5[x][y]==1) {goto nav1;}
        i5[x][y]=1; mas=mas-1;}
    if (cannel==5)
        {if (i6[x][y]==1) {goto nav1;}
        i6[x][y]=1; mas=mas-1;}

//change of mass and velocity in the cell - has to be zero
    if (mas!=0) {goto nav1;}
    velx=velx+(0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-0.5*i4[x][y]-i5[x][y]-
0.5*i6[x][y]);
    if (velx!=0) {goto nav2;}
    vely=vely+(sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y]);
    if (vely!=0) {goto nav2;}
return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationodd(void)
{
if (i1[x][y]==1)
    {
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
    nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]-0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x-1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
    nvy[x-1][y+1]=nvy[x-1][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }
}

```

```

if (i5[x][y]==1)
{
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]+0;}
}

if (i6[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationeven(void)
{
if (i1[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]+0;}
}

if (i3[x][y]==1)
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[x+1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
    nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle;}
}

if (i5[x][y]==1)

```

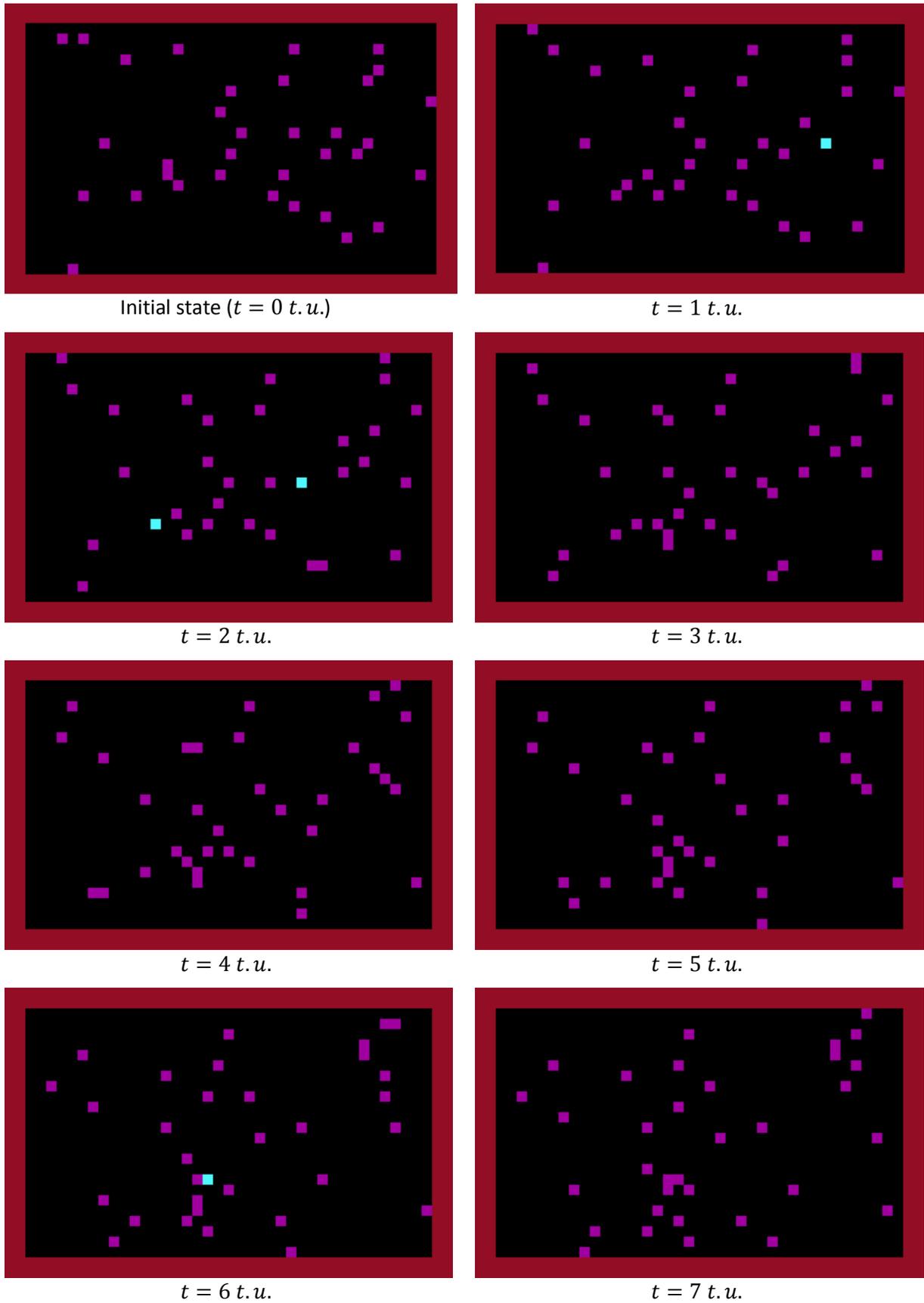
```
{
  if (nm[x+1][y]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
     nvy[x][y]=nvy[x][y]+0;}
  else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
        nvy[x+1][y]=nvy[x+1][y]-0;}
}

if (i6[x][y]==1)
  {
  if (nm[x+1][y-1]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
     nvy[x][y]=nvy[x][y]+sinangle;}
  else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
        nvy[x+1][y-1]=nvy[x+1][y-1]-sinangle;}
  }
return(0);
}
/*-----*/
```

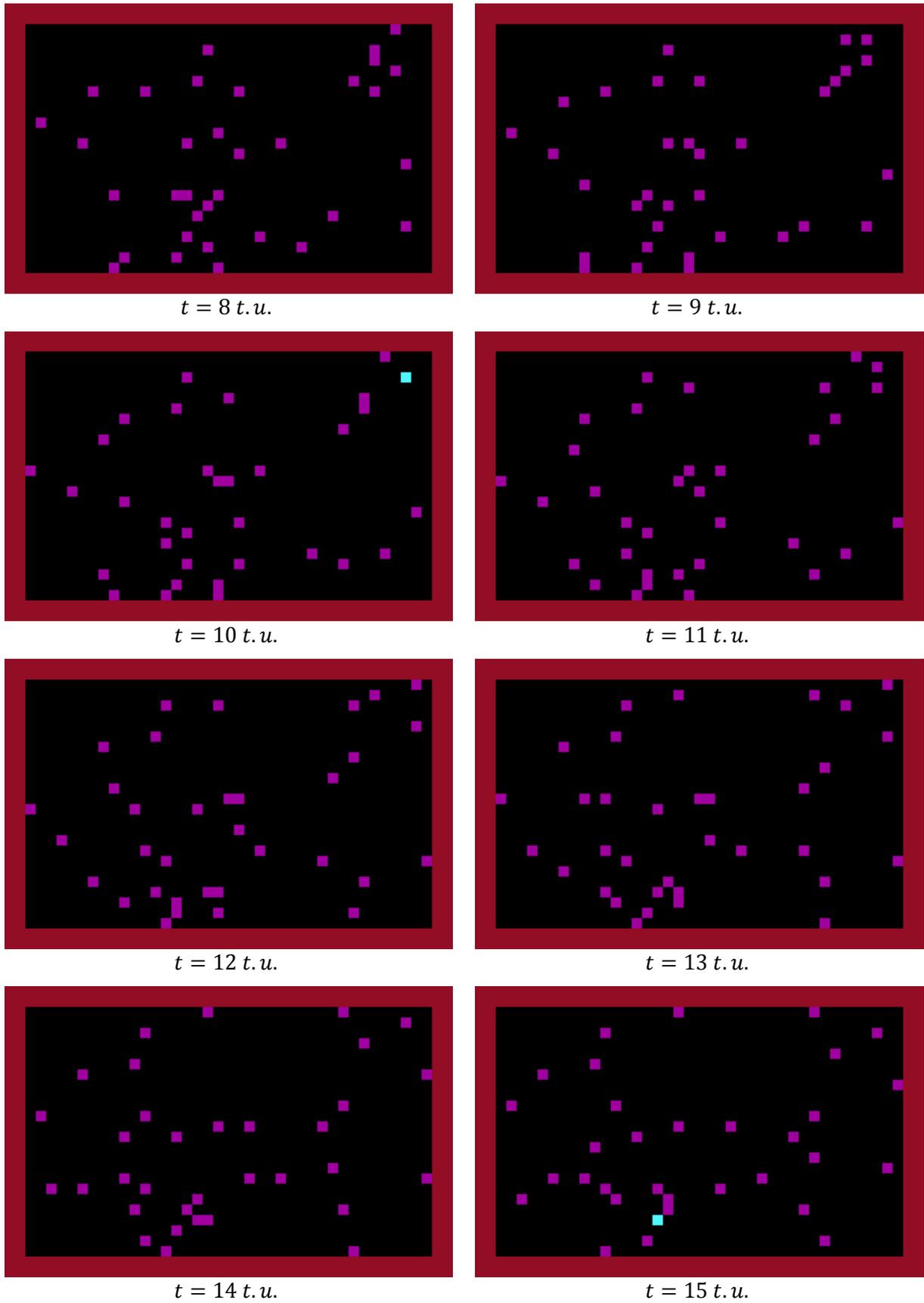
## APPENDIX D

# Time evolution of the FHP-1 Lattice Gas Cellular Automata model

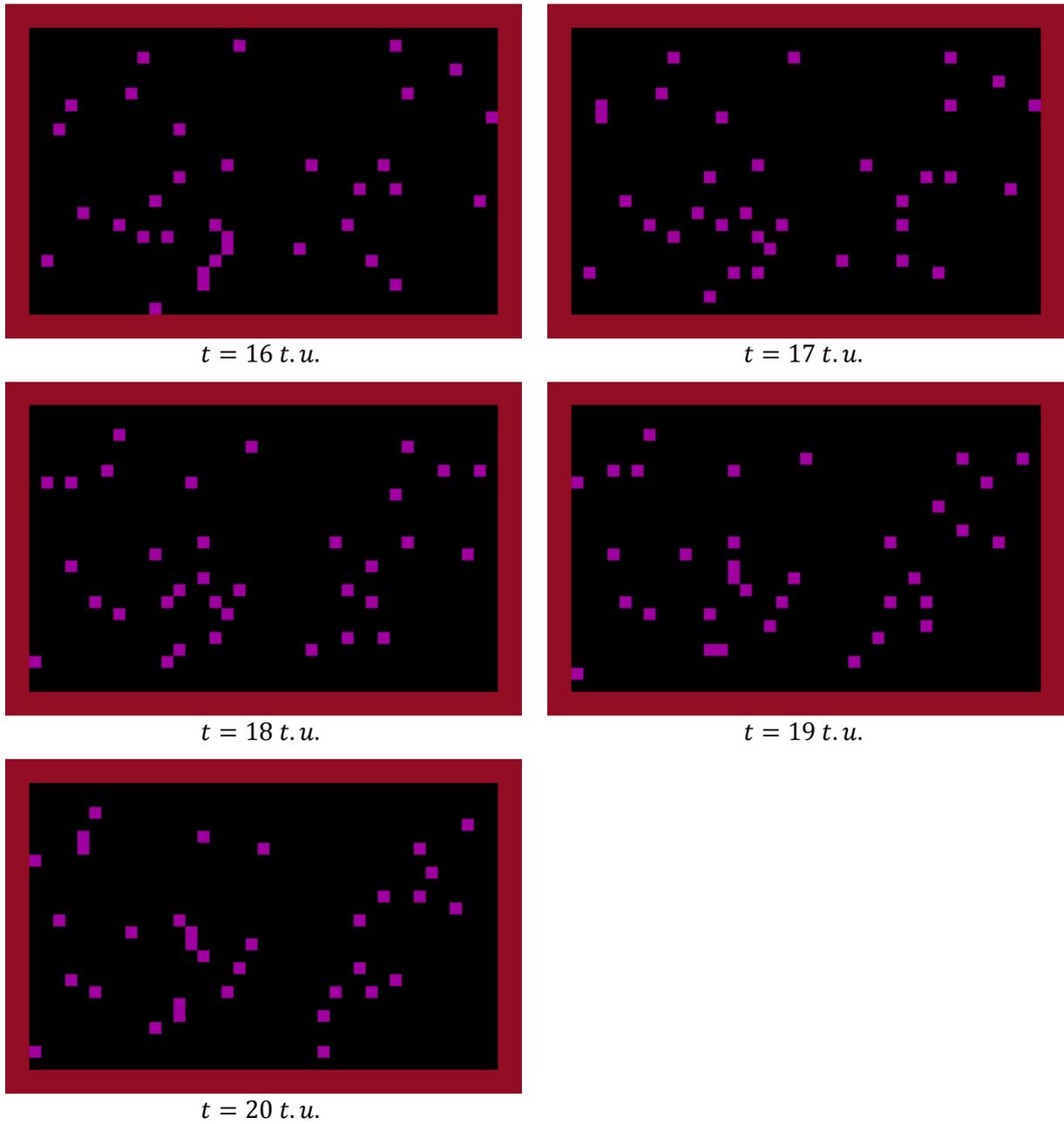
---



**Figure D-1:** Evolution of the developed FHP-1 LGCA model in time. Particle system is monitored for 20 time steps with an interval of 1 t.u. The size of the simulation domain is  $N_x \times N_y = 45 l.u. \times 25 l.u.$ , average lattice gas density is 0,2 m. u./l. u.



**Figure D-1 (continuation):** Evolution of the developed FHP-1 LGCA model in time. Particle system is monitored for 20 time steps with an interval of 1 t.u. The size of the simulation domain is  $N_x \times N_y = 45 \text{ l.u.} \times 25 \text{ l.u.}$ , average lattice gas density is 0,2 m.u./l.u.



**Figure D-1 (continuation):** Evolution of the developed FHP-1 LGCA model in time. Particle system is monitored for 20 time steps with an interval of 1 t.u. The size of the simulation domain is  $N_x \times N_y = 45 l.u. \times 25 l.u.$ , average lattice gas density is 0,2 m.u./l.u.

## APPENDIX E

# The FHP-1 Lattice Gas Cellular Automata algorithm for a Brownian motion simulation

---

```

//Basic FHP-1 LGCA algorithm - Brownian motion simulation

/* Code fragment 1: Header files and initialization of a simulation box */

//Definition of standard library functions
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <float.h>
# include <time.h>
# define DIRX 300
# define DIRY 300

//Declaration of main variables
int x, y, xmax=299, ymax=299;
float vx[DIRX][DIRY], nvx[DIRX][DIRY];
float vy[DIRX][DIRY], nvy[DIRX][DIRY];
int m[DIRX][DIRY], nm[DIRX][DIRY];
int i1[DIRX][DIRY], i2[DIRX][DIRY], i3[DIRX][DIRY], i4[DIRX][DIRY],
i5[DIRX][DIRY], i6[DIRX][DIRY];
float sinangle=0.866025403;
int pco=6;
int sig=15;
char str[25];
int I1, I2, I3, I4, I5, I6;
int cycle, cmax=20, series;
int i=25, print=4;
int fluid=3, boundary=4, hole=0, brownian=14;

//Declaration of variables of Brownian particle
int x1, y1, x2, y2;
int brownx, browny, collide=0;
int code1[DIRX][DIRY], code2[DIRX][DIRY], code3[DIRX][DIRY],
code4[DIRX][DIRY], code5[DIRX][DIRY], code6[DIRX][DIRY];
float distance=0;

//Declaration of subroutines
int collision(void);
int collisionbrown(void);
float propagationodd(void);
float propagationeven(void);
float propagationoddbrown(void);
float propagationevenbrown(void);

FILE *output0;

/*-----*/

/* Beginning of a main part of the program */
int main()
{

/* Code fragment 2: Graphic outputs setting */

int gdriver = DETECT, gmode, errorcode;

//initialize graphics and local variabls
initgraph (&gdriver, &gmode, "c:\\TC\\BGI");

//read result of initialization

```

```

        errorcode = graphresult();

        //an error occurred
        if (errorcode != grOk)
        {
            printf ("Graphics error: %s\n", grapherrormsg(errorcode));
            printf ("Press any key to halt:");
            getch();
            exit(1);
        }

/* Code fragment 3: Creation of the simulation domain and initial state of
the simulated system */

//Data arrays resetting
for (x=0; x<xmax+1; x++)
    {
        for (y=0; y<ymax+1; y++)
            {
                m[x][y]=0;
                nm[x][y]=0;
                vx[x][y]=0;
                nvx[x][y]=0;
                vy[x][y]=0;
                nvy[x][y]=0;
            }
    }

//Creation of solid boundaries of the simulation box
for (x=1; x<xmax; x++)
    {
        m[x][1]=7;
        m[x][2]=7;
        m[x][ymax-1]=7;
        m[x][ymax-2]=7;

        nm[x][1]=7;
        nm[x][2]=7;
        nm[x][ymax-1]=7;
        nm[x][ymax-2]=7;
        putpixel (x, 1, boundary);
        putpixel (x, ymax-1, boundary);
    }

for (y=1; y<ymax; y++)
    {
        m[1][y]=7;
        m[2][y]=7;
        m[xmax-1][y]=7;
        m[xmax-2][y]=7;
        nm[1][y]=7;
        nm[2][y]=7;
        nm[xmax-1][y]=7;
        nm[xmax-2][y]=7;
        putpixel (1, y, boundary);
        putpixel (xmax-1, y, boundary);
    }

randomize();

/* Code fragment 4: Occupation of cannels by fluid moving particles */

```

```

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
  {
    for (y=3; y<ymax-2; y=y+2)
      {
        if (m[x][y]!=7)
          {
            m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

            if (m[x-1][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

            if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}

            if (m[x-1][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

            if (m[x][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

            if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

            if (m[x][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

            //the total particles velocity in the node
            vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
            vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
            }}
      }

//even rows of the lattice
for (x=3; x<xmax-2; x++)
  {
    for (y=4; y<ymax-2; y=y+2)
      {
        if (m[x][y]!=7)
          {
            m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

            if (m[x][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else
{i1[x][y]=0;}

            if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else
{i2[x][y]=0;}

            if (m[x][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else
{i3[x][y]=0;}

            if (m[x+1][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else
{i4[x][y]=0;}

            if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else

```

```

{i5[x][y]=0;}

        if (m[x+1][y-1]<7) {I6=random(pco);}
        if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else
{i6[x][y]=0;}

        // the total particles velocity in the node
        vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
        vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
        }}
    }

//Determination of the Brownian particle
signal11:
brownx=random(xmax); browny=random(ymax);
if ((brownx<=xmax/2+i) && (brownx>=xmax/2-i))
    {if ((browny<=ymax/2+i) && (browny>=ymax/2-i))
        {if ((m[brownx][browny]>0) && (m[brownx][browny]<7))
            {m[brownx][browny]=m[brownx][browny]+13;}
        }
    else {goto signal11;}
}
else {goto signal11;}

/* Code fragment 5: Graphical and data outputs */

//Graphical outputs of the initial system configuration
for (x=1; x<xmax+1; x++)
    {
        for (y=1; y<ymax+1; y++)
            {
                putpixel (x, y, m[x][y]/print);
            }
    }

/* Code fragment 5-A: Data outputs */

//Opening the data file BROWN.CPP
if ((output0=fopen("C:\\Outputs\\Brownian\\brown00.cpp", "w"))==NULL)
    {
        printf("output file error\n");
        exit(0);
    }
fprintf (output0, "cycle, brownx, browny, x2, y2, distance\n");

/*-----*/

/* Code fragment 6: The main cycle of the algorithm */

for (cycle=0; cycle<cmax+1; cycle++)
{

/* Code fragment 6-A: Collision phase */

for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if ((m[x][y]>0) && (m[x][y]<7)) {collision();}
            if (m[x][y]>13) {collisionbrown();}
        }
    }
}

```

```

        if (m[x][y]>14) {collide=collide+1;}
    }
}

/* Code fragment 6-B: Propagation phase */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y=y+2)
    {
        if ((m[x][y]>0)&&(m[x][y]<7)) {propagationodd();}
        if (m[x][y]>13) {propagationoddbrown(); x1=x; y1=y;}
    }
}

//even rows of the lattice
for (x=3; x<xmax-2; x++)
{
    for (y=4; y<ymax-2; y=y+2)
    {
        if ((m[x][y]>0)&&(m[x][y]<7)) {propagationeven();}
        if (m[x][y]>13) {propagationevenbrown(); x1=x; y1=y;}
    }
}

/* Code fragment 7: Recording of a new sytem's state */

for (x=1; x<xmax+1; x++)
{
    for (y=1; y<ymax+1; y++)
    {
        m[x][y]=nm[x][y]; vx[x][y]=nvx[x][y]; vy[x][y]=nvy[x][y];
        if ((m[x][y]>0)&&(m[x][y]<7)) {putpixel (x, y, fluid);}
        if (m[x][y]==7) {putpixel (x, y, boundary);}
        if (m[x][y]==0) {putpixel (x, y, hole);}
        if (m[x][y]>13) {putpixel (x, y, brownian);}
    }
}

/* Code fragment 8: Data arrays resetting */

for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y++)
    {
        nm[x][y]=0; nvx[x][y]=0; nvy[x][y]=0;
        i1[x][y]=0; i2[x][y]=0; i3[x][y]=0;
        i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;
    }
}

/* Code fragment 9: Printout macro */
setfillstyle(1,0);
bar(getmaxx()-90,getmaxy()-90,getmaxx(),getmaxy()-70);
outtextxy(getmaxx()-120,getmaxy()-80,"cycle");
outtextxy(getmaxx()-70,getmaxy()-80,gcvt(series,sig,str));

//Outputs - BROWN.CPP
distance=sqrt(pow(x2-brownx,2)+pow(y2-browny,2));
fprintf(output0,"%8d %8d %8d %8d %8d %8.3f\n", series, brownx, browny, x2,
y2, distance);

```

```

//Collision with solid boundaries - the end of the simulation
if ((x2==3)|| (x2==xmax-3)) {goto signalend;}
if ((y2==3)|| (y2==ymax-3)) {goto signalend;}

series=series+1;
getch();
} //The end of the main cycle

signalend:

/* Code fragment 10: Final operations */

getch();
closegraph();
fclose (output0);
return (0);
} //The end of the main part of the algorithm

/*-----*/

//SUBROUTINES

/*-----*/
//Collision phase - fluid
int collision(void)
{
int cannel=0;
int mas=0;
float velx=0;
float vely=0;

signal2:
velx=vx[x][y]; vely=vy[x][y]; mas=m[x][y];
i1[x][y]=0; i2[x][y]=0; i3[x][y]=0; i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;

signal1:
cannel=0;
cannel=random(6);

if (cannel==0)
    {if (i1[x][y]==1) {goto signal1;}
    i1[x][y]=1; mas=mas-1;}
if (cannel==1)
    {if (i2[x][y]==1) {goto signal1;}
    i2[x][y]=1; mas=mas-1;}
if (cannel==2)
    {if (i3[x][y]==1) {goto signal1;}
    i3[x][y]=1; mas=mas-1;}
if (cannel==3)
    {if (i4[x][y]==1) {goto signal1;}
    i4[x][y]=1; mas=mas-1;}
if (cannel==4)
    {if (i5[x][y]==1) {goto signal1;}
    i5[x][y]=1; mas=mas-1;}
if (cannel==5)
    {if (i6[x][y]==1) {goto signal1;}
    i6[x][y]=1; mas=mas-1;}

//Change of mass and velocity in the node - has to be zero
if (mas!=0) {goto signal1;}

```

```

        velx=velx+(0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-0.5*i4[x][y]-i5[x][y]-
0.5*i6[x][y]);
        if (velx!=0) {goto signal2;}
        vely=vely+(sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y]);
        if (vely!=0) {goto signal2;}
return(0);
}

/*-----*/

//Collision phase - Brownian particle
int collisionbrown(void)
{
int cannel=0;
int mas=0;
int brownp=0;
float velx=0;
float vely=0;

signal2:
velx=vx[x][y]; vely=vy[x][y]; mas=m[x][y]-13;
i1[x][y]=0; i2[x][y]=0; i3[x][y]=0; i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;
code1[x][y]=0; code2[x][y]=0; code3[x][y]=0; code4[x][y]=0; code5[x][y]=0;
code6[x][y]=0;

signal1:
cannel=0;
cannel=random(6);

    if (cannel==0)
        {if (i1[x][y]==1) {goto signal1;}
        i1[x][y]=1; mas=mas-1;}
    if (cannel==1)
        {if (i2[x][y]==1) {goto signal1;}
        i2[x][y]=1; mas=mas-1;}
    if (cannel==2)
        {if (i3[x][y]==1) {goto signal1;}
        i3[x][y]=1; mas=mas-1;}
    if (cannel==3)
        {if (i4[x][y]==1) {goto signal1;}
        i4[x][y]=1; mas=mas-1;}
    if (cannel==4)
        {if (i5[x][y]==1) {goto signal1;}
        i5[x][y]=1; mas=mas-1;}
    if (cannel==5)
        {if (i6[x][y]==1) {goto signal1;}
        i6[x][y]=1; mas=mas-1;}

//Change of mass and velocity in the node - has to be zero
    if (mas!=0) {goto signal1;}
    velx=velx+(0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-0.5*i4[x][y]-i5[x][y]-
0.5*i6[x][y]);
    if (velx!=0) {goto signal2;}
    vely=vely+(sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y]);
    if (vely!=0) {goto signal2;}

//Marking of the Brownian particle
signal111:
brownp=random(6);

```

```

if ((brownp==0)&&(i1[x][y]==1)){i1[x][y]=13; code1[x][y]=13; goto
signal112;} else {code1[x][y]=0;}
if ((brownp==1)&&(i2[x][y]==1)){i2[x][y]=13; code2[x][y]=13; goto
signal112;} else {code2[x][y]=0;}
if ((brownp==2)&&(i3[x][y]==1)){i3[x][y]=13; code3[x][y]=13; goto
signal112;} else {code3[x][y]=0;}
if ((brownp==3)&&(i4[x][y]==1)){i4[x][y]=13; code4[x][y]=13; goto
signal112;} else {code4[x][y]=0;}
if ((brownp==4)&&(i5[x][y]==1)){i5[x][y]=13; code5[x][y]=13; goto
signal112;} else {code5[x][y]=0;}
if ((brownp==5)&&(i6[x][y]==1)){i6[x][y]=13; code6[x][y]=13; goto
signal112;} else {code6[x][y]=0; goto signal111;}
signal112:

return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)-
fluid
float propagationodd(void)
{
if (i1[x][y]==1)
{
if (nm[x-1][y-1]==7)
{nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
nvy[x][y]=nvy[x][y]+sinangle;}
else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
if (nm[x-1][y]==7)
{nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
nvy[x][y]=nvy[x][y]+0;}
else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
nvy[x-1][y]=nvy[x-1][y]-0;}
}

if (i3[x][y]==1)
{
if (nm[x-1][y+1]==7)
{nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
nvy[x][y]=nvy[x][y]-sinangle;}
else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
nvy[x-1][y+1]=nvy[x-1][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
if (nm[x][y+1]==7)
{nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
nvy[x][y]=nvy[x][y]-sinangle;}
else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
if (nm[x+1][y]==7)

```

```

        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
          nvy[x][y]=nvy[x][y]-0;}
else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
      nvy[x+1][y]=nvy[x+1][y]+0;}
}

if (i6[x][y]==1)
{
  if (nm[x][y-1]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
      nvy[x][y]=nvy[x][y]+sinangle;}
  else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
        nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection) -
fluid

float propagationeven(void)
{
if (i1[x][y]==1)
  {
  if (nm[x][y-1]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
      nvy[x][y]=nvy[x][y]+sinangle;}
  else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
        nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}

if (i2[x][y]==1)
  {
  if (nm[x-1][y]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
      nvy[x][y]=nvy[x][y]-0;}
  else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
        nvy[x-1][y]=nvy[x-1][y]+0;}
}

if (i3[x][y]==1)
  {
  if (nm[x][y+1]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
      nvy[x][y]=nvy[x][y]-sinangle;}
  else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
        nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i4[x][y]==1)
  {
  if (nm[x+1][y+1]==7)
    {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
      nvy[x][y]=nvy[x][y]-sinangle;}
  else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
        nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle;}
}

if (i5[x][y]==1)
  {

```

```

    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]+0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvx[x+1][y]=nvx[x+1][y]-0;}
}

if (i6[x][y]==1)
{
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
    nvx[x+1][y-1]=nvx[x+1][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)-
Brownian particle

float propagationoddbrown(void)
{
if ((i1[x][y]==13) || (i1[x][y]==1))
    {
        if (nm[x-1][y-1]==7)
            {nm[x][y]=nm[x][y]+1+code1[x][y]; nvx[x][y]=nvx[x][y]+0.5;
            nvx[x][y]=nvx[x][y]+sinangle; x2=x; y2=y;}
        else {nm[x-1][y-1]=nm[x-1][y-1]+1+code1[x][y]; nvx[x-1][y-1]=nvx[x-
1][y-1]-0.5;
        nvx[x-1][y-1]=nvx[x-1][y-1]-sinangle; x2=x-1; y2=y-1;}
    }

if ((i2[x][y]==13) || (i2[x][y]==1))
    {
        if (nm[x-1][y]==7)
            {nm[x][y]=nm[x][y]+1+code2[x][y]; nvx[x][y]=nvx[x][y]+1;
            nvx[x][y]=nvx[x][y]+0; x2=x; y2=y;}
        else {nm[x-1][y]=nm[x-1][y]+1+code2[x][y]; nvx[x-1][y]=nvx[x-1][y]-1;
        nvx[x-1][y]=nvx[x-1][y]-0; x2=x-1; y2=y;}
    }

if ((i3[x][y]==13) || (i3[x][y]==1))
    {
        if (nm[x-1][y+1]==7)
            {nm[x][y]=nm[x][y]+1+code3[x][y]; nvx[x][y]=nvx[x][y]+0.5;
            nvx[x][y]=nvx[x][y]-sinangle; x2=x; y2=y;}
        else {nm[x-1][y+1]=nm[x-1][y+1]+1+code3[x][y]; nvx[x-1][y+1]=nvx[x-
1][y+1]-0.5;
        nvx[x-1][y+1]=nvx[x-1][y+1]+sinangle; x2=x-1; y2=y+1;}
    }

if ((i4[x][y]==13) || (i4[x][y]==1))
    {
        if (nm[x][y+1]==7)
            {nm[x][y]=nm[x][y]+1+code4[x][y]; nvx[x][y]=nvx[x][y]-0.5;
            nvx[x][y]=nvx[x][y]-sinangle; x2=x; y2=y;}
        else {nm[x][y+1]=nm[x][y+1]+1+code4[x][y];
nvx[x][y+1]=nvx[x][y+1]+0.5;
        nvx[x][y+1]=nvx[x][y+1]+sinangle; x2=x; y2=y+1;}
    }
}

```

```

    }

if ((i5[x][y]==13) || (i5[x][y]==1))
{
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1+code5[x][y]; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]-0; x2=x; y2=y;}
    else {nm[x+1][y]=nm[x+1][y]+1+code5[x][y]; nvx[x+1][y]=nvx[x+1][y]+1;
    nvx[x+1][y]=nvx[x+1][y]+0; x2=x+1; y2=y;}
}

if ((i6[x][y]==13) || (i6[x][y]==1))
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1+code6[x][y]; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle; x2=x; y2=y;}
    else {nm[x][y-1]=nm[x][y-1]+1+code6[x][y]; nvx[x][y-1]=nvx[x][y-
1]+0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle; x2=x; y2=y-1;}
}
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection) -
Brownian particle

float propagationevenbrown(void)
{
if ((i1[x][y]==13) || (i1[x][y]==1))
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1+code1[x][y]; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]+sinangle; x2=x; y2=y;}
    else {nm[x][y-1]=nm[x][y-1]+1+code1[x][y]; nvx[x][y-1]=nvx[x][y-1]-
0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle; x2=x; y2=y-1;}
}

if ((i2[x][y]==13) || (i2[x][y]==1))
{
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1+code2[x][y]; nvx[x][y]=nvx[x][y]+1;
        nvx[x][y]=nvx[x][y]-0; x2=x; y2=y;}
    else {nm[x-1][y]=nm[x-1][y]+1+code2[x][y]; nvx[x-1][y]=nvx[x-1][y]-1;
    nvx[x-1][y]=nvx[x-1][y]+0; x2=x-1; y2=y;}
}

if ((i3[x][y]==13) || (i3[x][y]==1))
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1+code3[x][y]; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]-sinangle; x2=x; y2=y;}
    else {nm[x][y+1]=nm[x][y+1]+1+code3[x][y]; nvx[x][y+1]=nvx[x][y+1]-
0.5;
    nvx[x][y+1]=nvx[x][y+1]+sinangle; x2=x; y2=y+1;}
}

if ((i4[x][y]==13) || (i4[x][y]==1))
{
    if (nm[x+1][y+1]==7)

```

```

        {nm[x][y]=nm[x][y]+1+code4[x][y]; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle; x2=x; y2=y;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1+code4[x][y];
nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
        nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle; x2=x+1; y2=y+1;}
    }

if ((i5[x][y]==13) || (i5[x][y]==1))
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1+code5[x][y]; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]+0; x2=x; y2=y;}
    else {nm[x+1][y]=nm[x+1][y]+1+code5[x][y]; nvx[x+1][y]=nvx[x+1][y]+1;
        nvy[x+1][y]=nvy[x+1][y]-0; x2=x+1; y2=y;}
    }

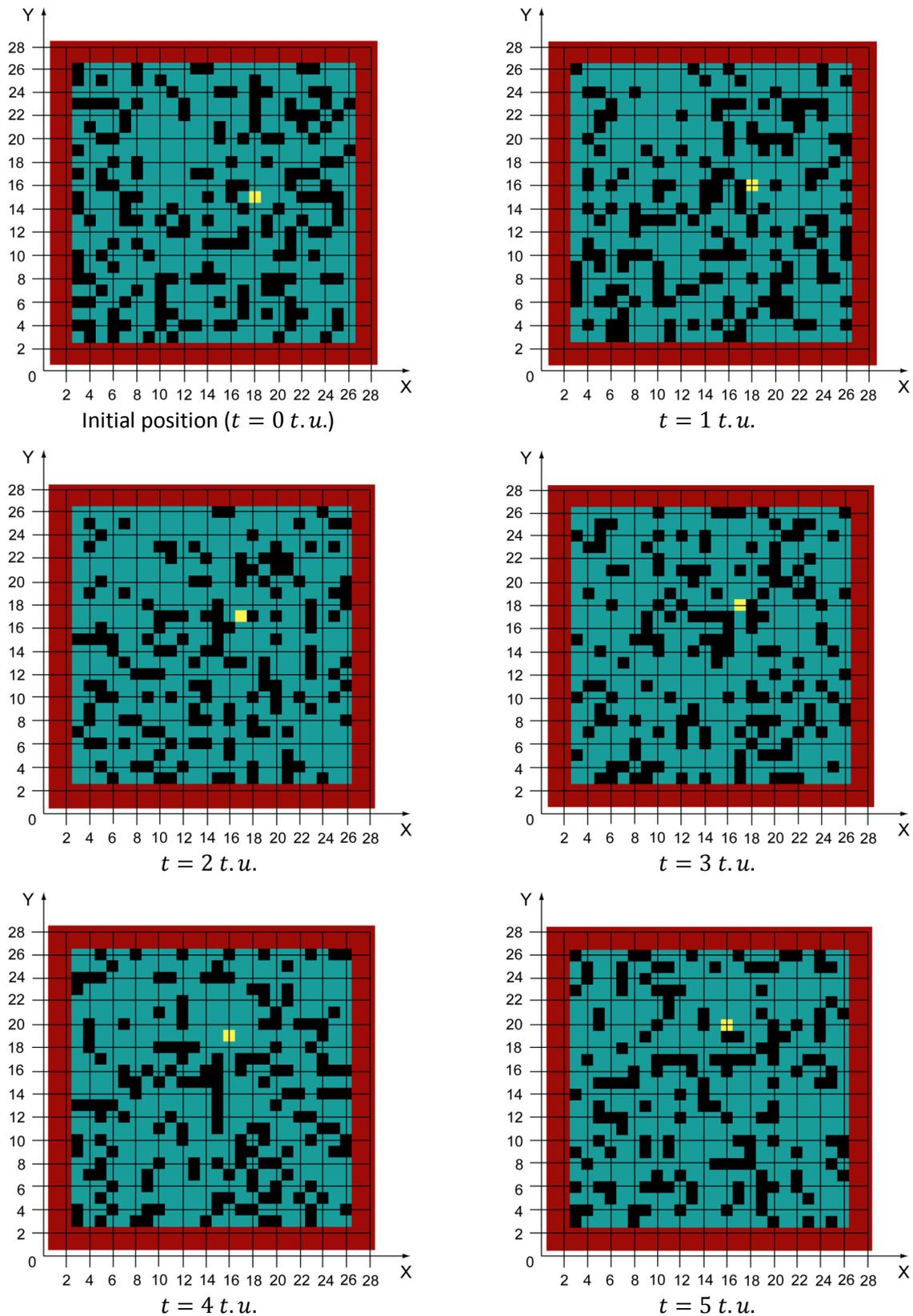
if ((i6[x][y]==13) || (i6[x][y]==1))
    {
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1+code6[x][y]; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle; x2=x; y2=y;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1+code6[x][y]; nvx[x+1][y-
1]=nvx[x+1][y-1]+0.5;
        nvy[x+1][y-1]=nvy[x+1][y-1]-sinangle; x2=x+1; y2=y-1;}
    }
return(0);
}
/*-----*/

```

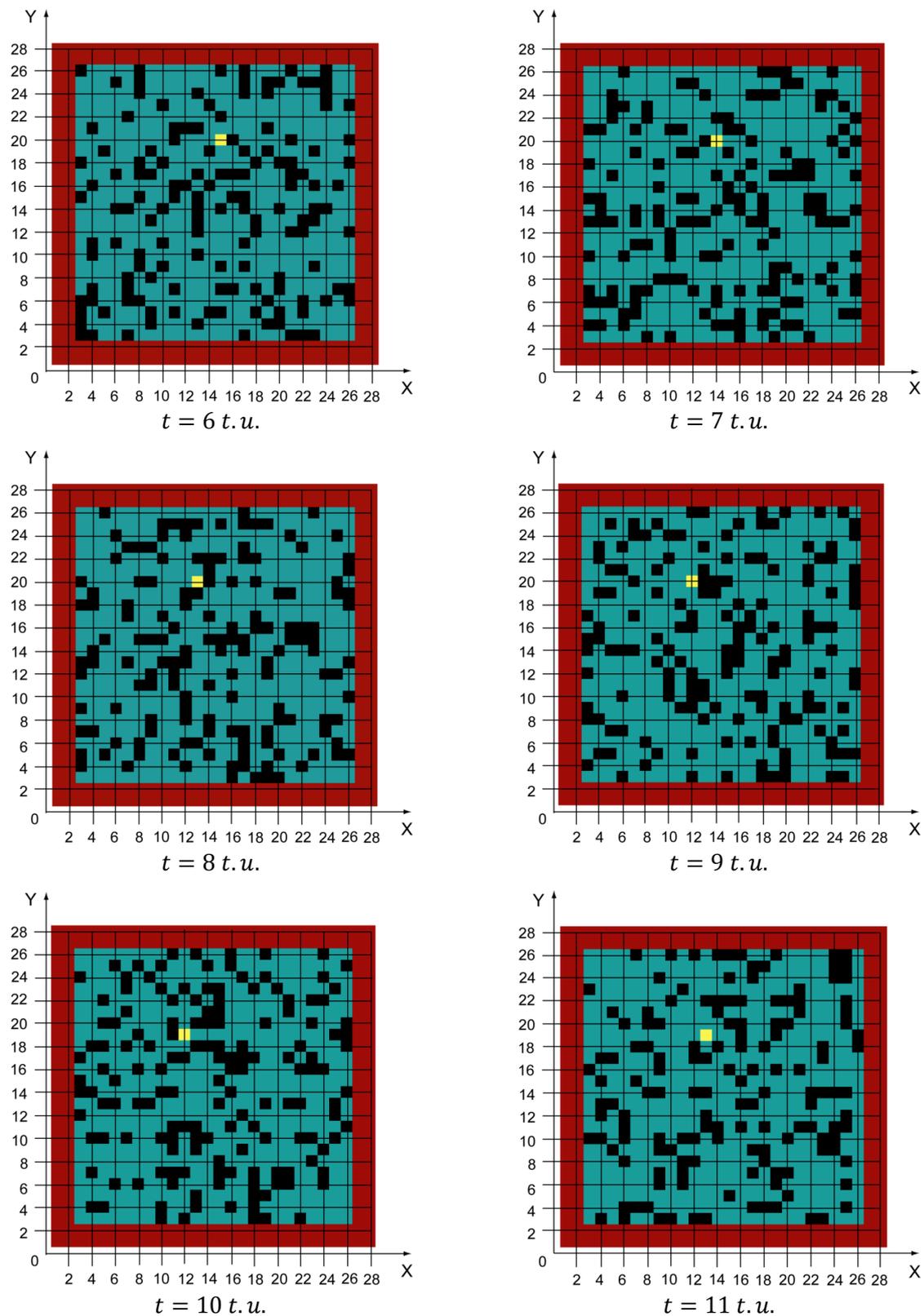
## APPENDIX F

Computer simulation of the Brownian motion.  
Evolution in time for 20 time steps

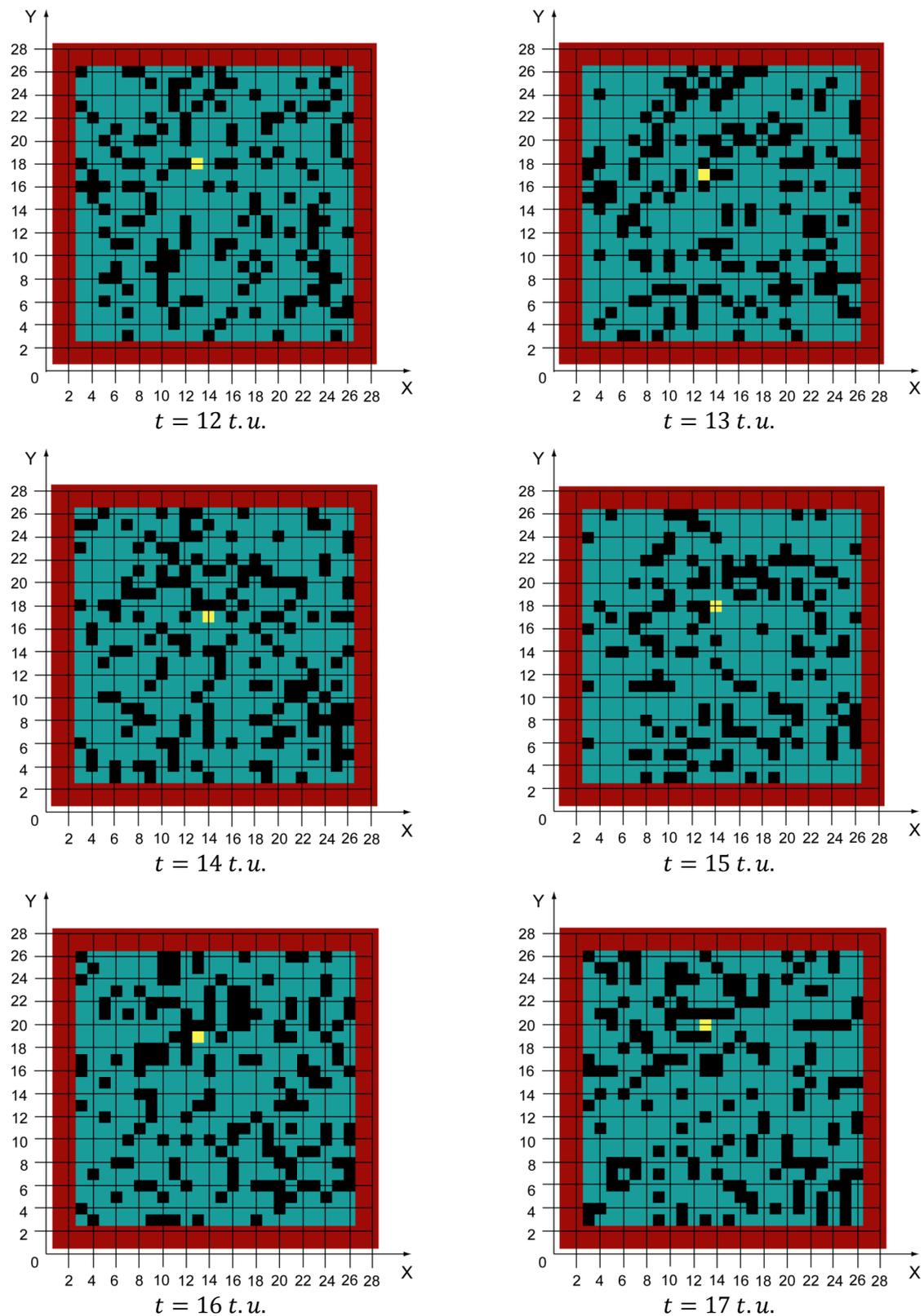
---



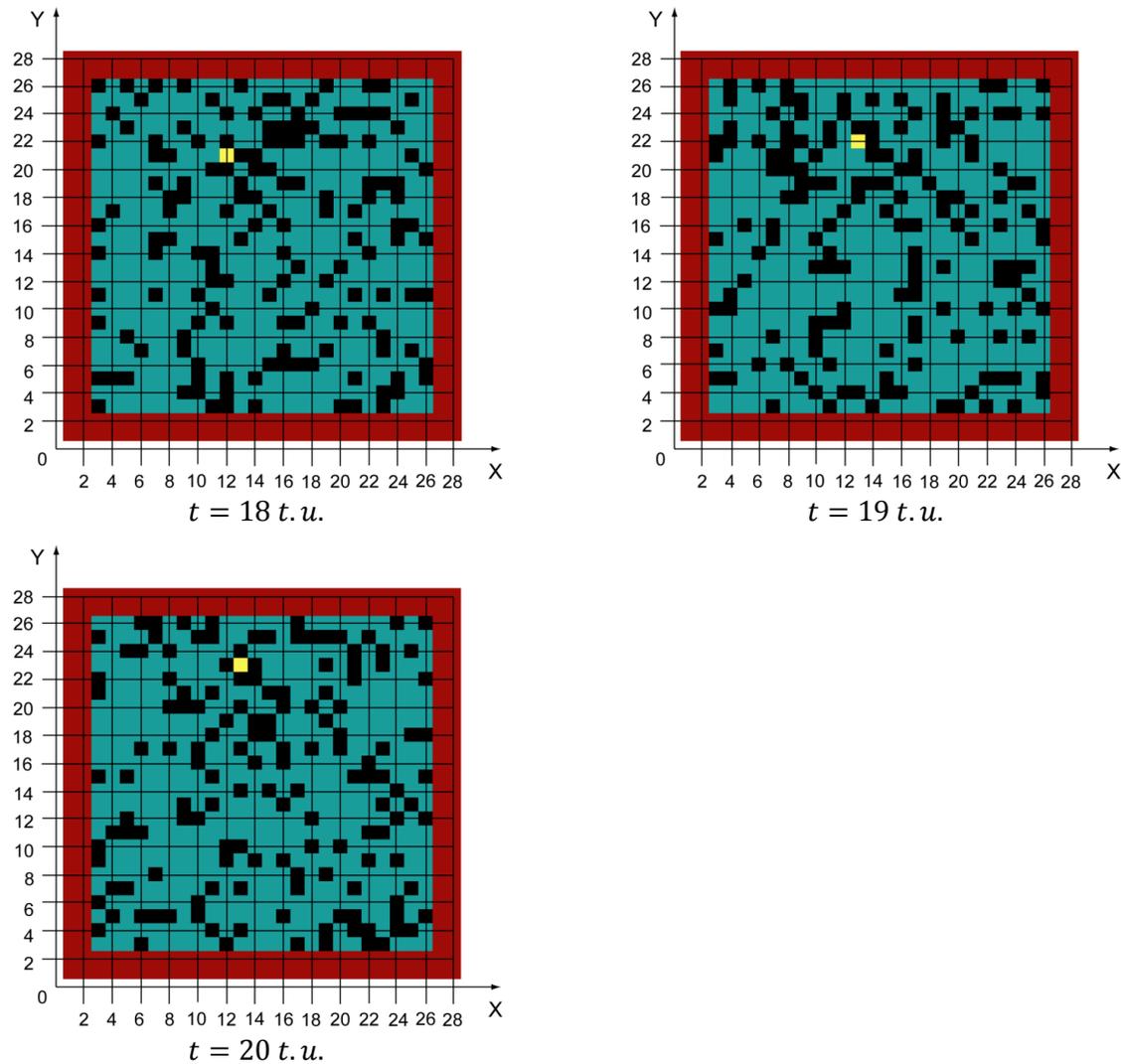
**Figure F-1:** Simulation of the Brownian motion presented on the reduced simulation domain of a size  $N_x \times N_y = 30 \text{ l.u.} \times 30 \text{ l.u.}$ . The lattice gas average density is  $3 \text{ m.u./l.u.}$ . The simulation domain is bounded by solid walls (red lines), blue regions corresponds with moving particles, black squares present empty lattice nodes. The Brownian particle is yellow one. It is monitored for 20 time steps with an interval of  $1 t.u.$ . The developed FHP-1 LGCA is used for the simulation



**Figure F-1 (continuation):** Simulation of the Brownian motion presented on the reduced simulation domain of a size  $N_x \times N_y = 30 \text{ l.u.} \times 30 \text{ l.u.}$ . The lattice gas average density is  $3 \text{ m.u./l.u.}$ . The simulation domain is bounded by solid walls (red lines), blue regions corresponds with moving particles, black squares present empty lattice nodes. The Brownian particle is yellow one. It is monitored for 20 time steps with an interval of  $1 \text{ t.u.}$ . The developed FHP-1 LGCA is used for the simulation



**Figure F-1 (continuation):** Simulation of the Brownian motion presented on the reduced simulation domain of a size  $N_x \times N_y = 30 \text{ l.u.} \times 30 \text{ l.u.}$ . The lattice gas average density is  $3 \text{ m.u./l.u.}$ . The simulation domain is bounded by solid walls (red lines), blue regions corresponds with moving particles, black squares present empty lattice nodes. The Brownian particle is yellow one. It is monitored for 20 time steps with an interval of  $1 \text{ t.u.}$ . The developed FHP-1 LGCA is used for the simulation

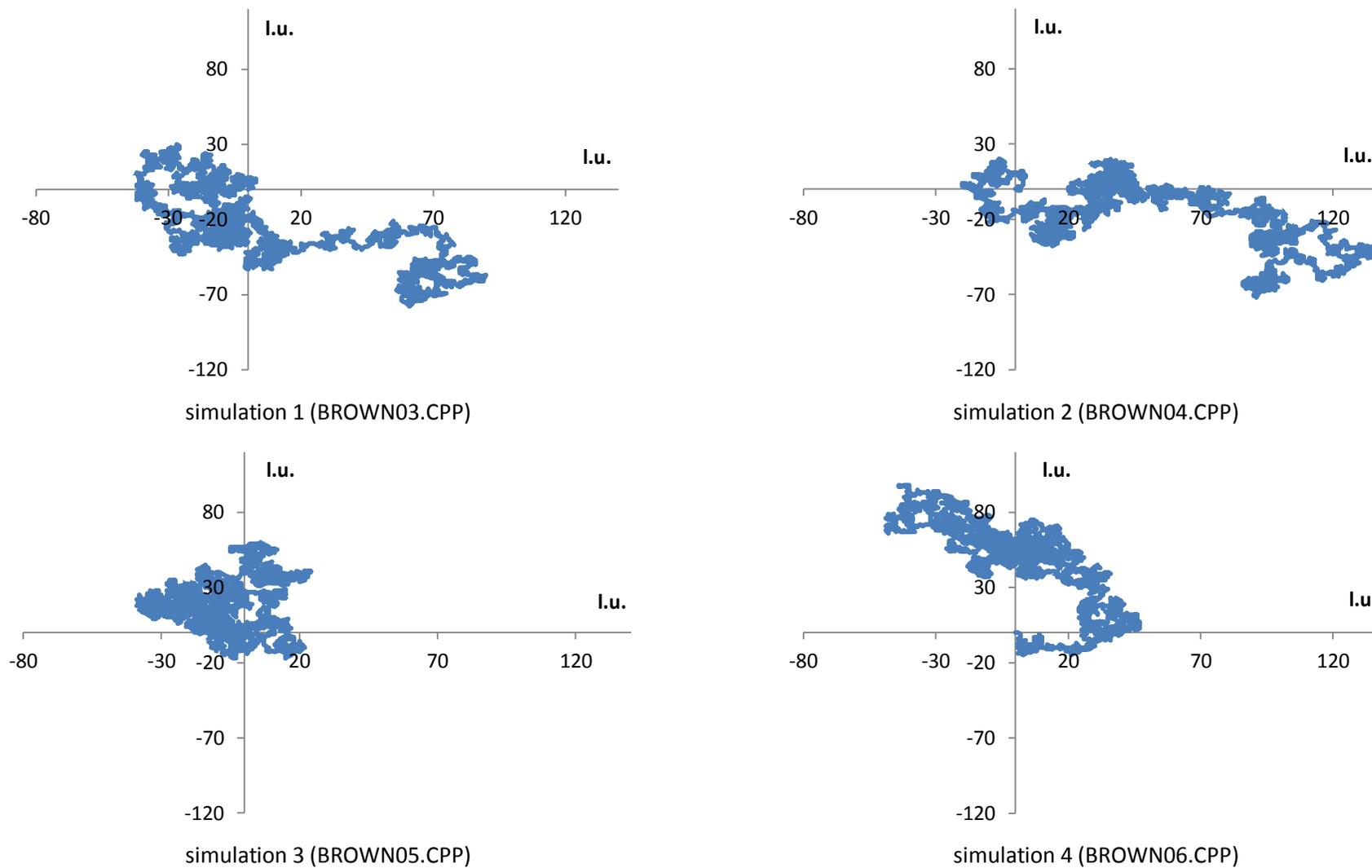


**Figure F-1 (continuation):** Simulation of the Brownian motion presented on the reduced simulation domain of a size  $N_x \times N_y = 30 l.u. \times 30 l.u.$  The lattice gas average density is  $3 m.u./l.u.$  The simulation domain is bounded by solid walls (red lines), blue regions corresponds with moving particles, black squares present empty lattice nodes. The Brownian particle is yellow one. It is monitored for 20 time steps with an interval of  $1 t.u.$  The developed FHP-1 LGCA is used for the simulation

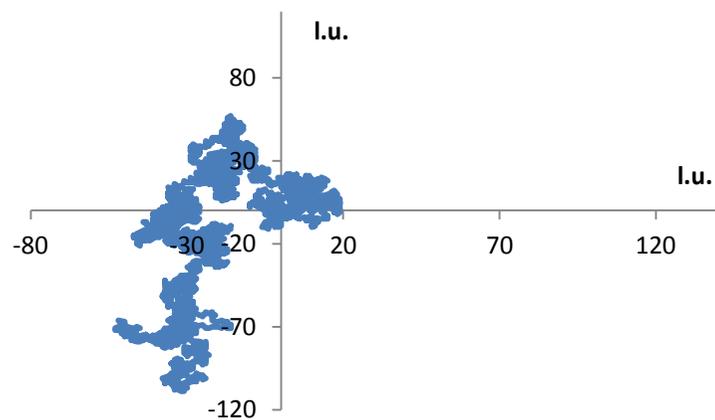
**APPENDIX G**

Computer simulation of the Brownian motion.  
Paths of the Brownian particle after 4000 time  
steps. Experiment 1

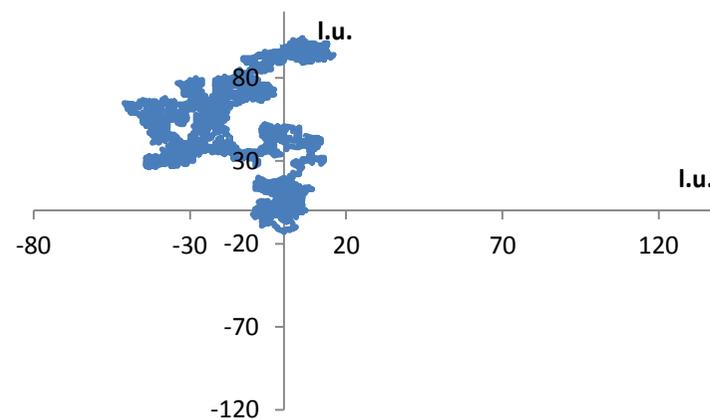
---



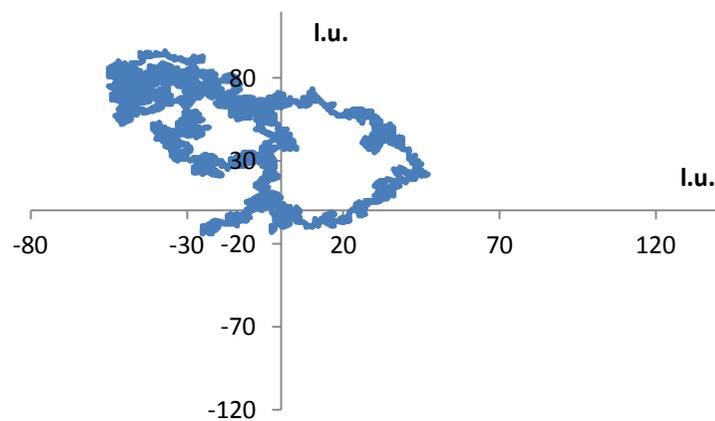
**Figure G-1:** Paths of the Brownian particle after 4000 time steps. The size of the simulation domain is  $N_x \times N_y = 300 \text{ l.u.} \times 300 \text{ l.u.}$  Lattice gas average density is  $3 \text{ m.u./l.u.}$



simulation 5 (BROWN08.CPP)



simulation 6 (BROWN11.CPP)



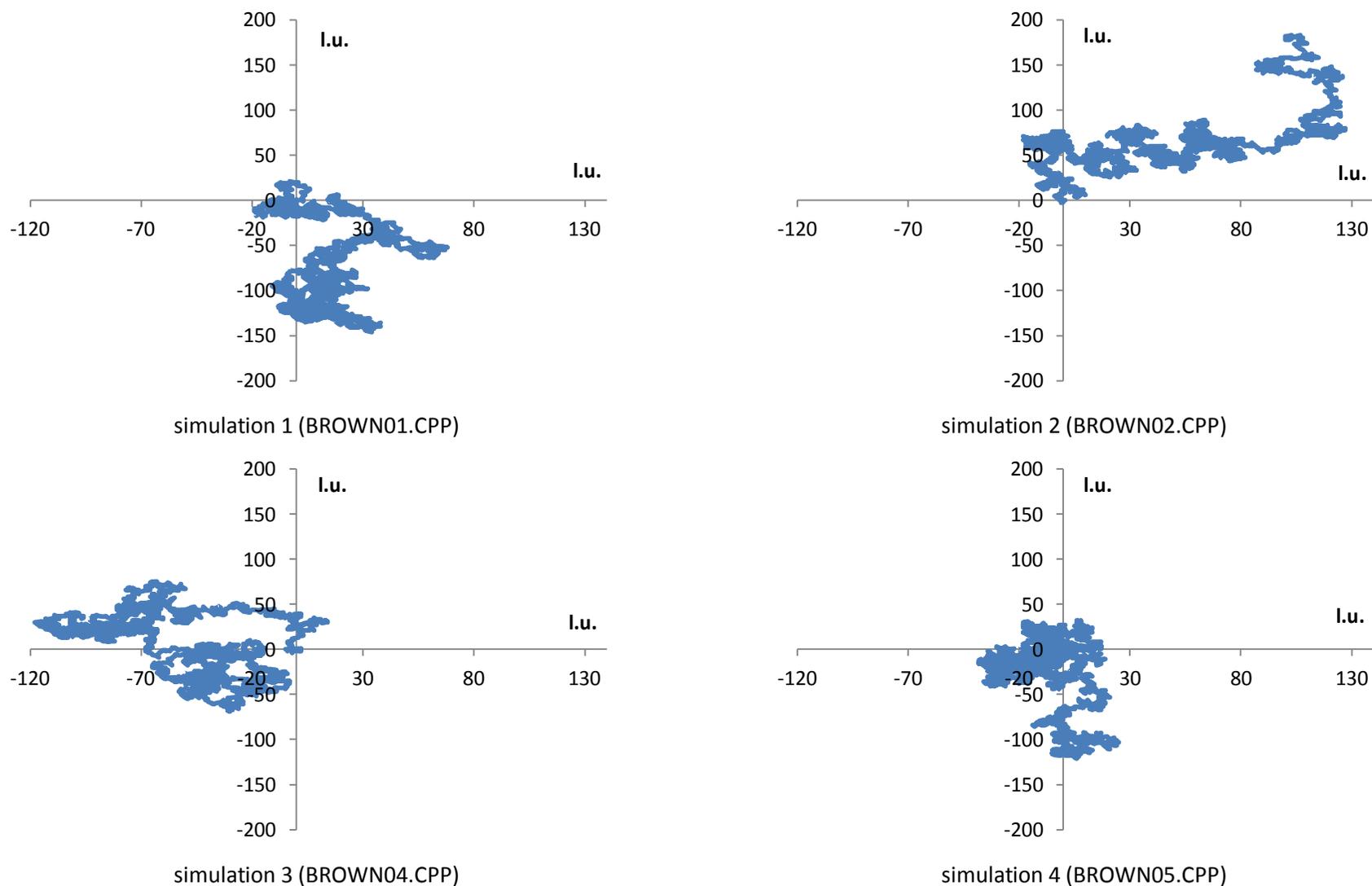
simulation 7 (BROWN12.CPP)

**Figure G-1 (continuation):** Paths of the Brownian particle after 4000 time steps. The size of the simulation domain is  $N_x \times N_y = 300 \text{ l.u.} \times 300 \text{ l.u.}$  Lattice gas average density is  $3 \text{ m.u./l.u.}$

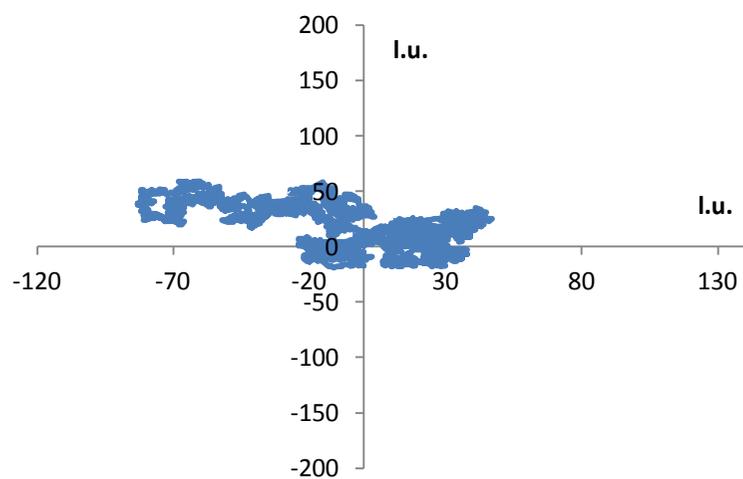
**APPENDIX H**

Computer simulation of the Brownian motion.  
Paths of the Brownian particle after 4000 time  
steps. Experiment 2

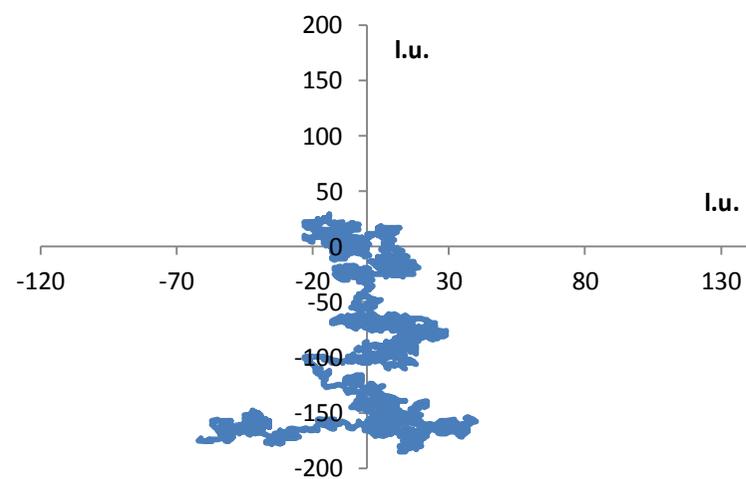
---



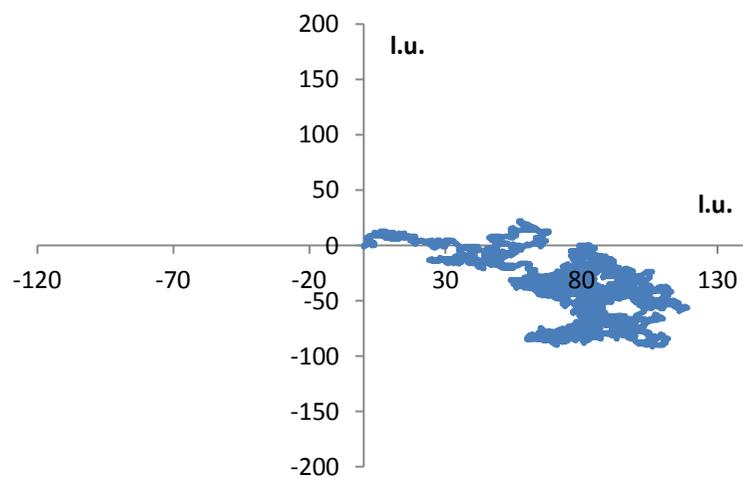
**Figure H-:** Paths of the Brownian particle after 4000 time steps. The size of the simulation domain is  $N_x \times N_y = 400 \text{ l.u.} \times 400 \text{ l.u.}$  Lattice gas average density is  $1,5 \text{ m.u./l.u.}$



simulation 5 (BROWN07.CPP)



simulation 6 (BROWN08.CPP)



simulation 7 (BROWN09.CPP)

**Figure H-1 (continuation):** Paths of the Brownian particle after 4000 time steps. The size of the simulation domain is  $N_x \times N_y = 400 \text{ l.u.} \times 400 \text{ l.u.}$  Lattice gas average density is  $1,5 \text{ m.u./l.u.}$

## APPENDIX I

# The FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow simulation

---

```

//FHP-1 LGCA for Poiseuille flow simulation

/* Code fragment 1: Header files and initialization of the simulation box
*/

//Definition of standard library functions
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <float.h>
# include <time.h>
# define DIRX 550
# define DIRY 200

//Declaration of variables
int x, y, xmax=549, ymax=199;
float vx[DIRX][DIRY], nvx[DIRX][DIRY];
float vy[DIRX][DIRY], nvy[DIRX][DIRY];
int m[DIRX][DIRY], nm[DIRX][DIRY];
int i1[DIRX][DIRY], i2[DIRX][DIRY], i3[DIRX][DIRY], i4[DIRX][DIRY],
i5[DIRX][DIRY], i6[DIRX][DIRY];
int mass=0, node=0;
float velocity=0;
int ventilator=5, force=1;
float sinangle=0.866025403, print=5.5, step=0;
float flow, V[DIRY];
int pco=3;
int transfer14=0, transfer25=0, transfer36=0;
int sig=15;
char str[25];
int I1, I2, I3, I4, I5, I6;
int cycle, cmax=10000, series;

//Declaration of subroutines
int collision(void);
float propagationodd(void);
float propagationeven(void);
float propagationleftsideodd(void);
float propagationleftsideeven(void);
float propagationrightsideodd(void);
float propagationrightsideeven(void);
float turnright(void);
float profile(void);
FILE *output0;
FILE *output1;
/*-----*/
/* Beginning of a main part of the program */
int main()
{

/* Code fragment 2: Graphic outputs setting */

int gdriver = DETECT, gmode, errorcode;

//initialize graphics and local variabls
initgraph (&gdriver, &gmode, "c:\\TC\\BGI");

//read result of initialization
errorcode = graphresult();

```

```

//an error occurred
if (errorcode != grOk)
{
    printf ("Graphics error: %s\n", grapherrormsg(errorcode));
    printf ("Press any key to halt:");
    getch();
    exit(1);
}

/* Code fragment 3: Creation of the simulation domain and initial state of
the simulated system */

//Data arrays resetting
for (x=0; x<xmax+1; x++)
{
    for (y=0; y<ymax+1; y++)
    {
        m[x][y]=0;
        nm[x][y]=0;
        vx[x][y]=0;
        nvx[x][y]=0;
        vy[x][y]=0;
        nvy[x][y]=0;
    }
}

//Creation of solid boundaries of the simulation box
for (x=1; x<xmax; x++)
{
    m[x][1]=7;
    m[x][2]=7;
    m[x][ymax-1]=7;
    m[x][ymax-2]=7;

    nm[x][1]=7;
    nm[x][2]=7;
    nm[x][ymax-1]=7;
    nm[x][ymax-2]=7;
    putpixel (x, 1, m[x][1]*print);
    putpixel (x, ymax-1, m[x][ymax-1]*print);
}

//randomize();

/* Code fragment 4: Occupation of cannels by fluid moving particles */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y=y+2)
    {
        if (m[x][y]!=7)
        {
            m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

            if (m[x-1][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

            if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}

            if (m[x-1][y+1]<7) {I3=random(pco);}

```

```

if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

        if (m[x][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

        if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

        if (m[x][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

        //the total particles velocity in the node
        vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
        vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
        }}
    }

//even rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=4; y<ymax-2; y=y+2)
            {
                if (m[x][y]!=7)
                    {
                        m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

                        if (m[x][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

                        if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}

                        if (m[x][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

                        if (m[x+1][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

                        if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

                        if (m[x+1][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

                        // the total particles velocity in the node
                        vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
                        vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
                        }}
            }

/* Code fragment 5: Graphical outputs of the initial system configuration
*/

for (x=1; x<xmax+1; x++)
    {
        for (y=1; y<ymax+1; y++)
            {
                putpixel (x, y, m[x][y]*print);
            }
    }

```

```

    }
}

//Opening the data file FLOW.CPP
if ((output0=fopen("C:\\Outputs\\Poiseuil\\Flow02.cpp","w"))==NULL)
{
    printf("output file error\n");
    exit(0);
}
fprintf (output0, "trasfer14, transfer25, transfer36, cycle, node, mass,
flow\n");
/*-----*/

/* Code fragment 6: The main cycle of the algorithm */

for (cycle=0; cycle<cmax+1; cycle++)
{

/* Code fragment 6-A: Collision phase */

for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if ((m[x][y]>0)&&(m[x][y]!=7)) {collision();}
        }
    }

/* Code fragment 6-B: Pressure gradient */

transfer14=0; transfer25=0; transfer36=0;
for (x=3; x<ventilator; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if ((m[x][y]>0)&&(m[x][y]!=7))
                { if (random(force)<100) {turnright();}}
        }
    }

/* Code fragment 6-C: Propagation phase */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=3; y<ymax-2; y=y+2)
            {
                if ((m[x][y]>0)&&(m[x][y]!=7))
                    {
                        if (x==3) {propagationleftsideodd();}
                        if (x==xmax-3) {propagationrightsideodd();}
                        if ((x>3)&&(x<xmax-3)) {propagationodd();}
                    }
            }
    }

//even rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=4; y<ymax-2; y=y+2)
            {
                if ((m[x][y]>0)&&(m[x][y]!=7))
                    {
                        if (x==3) {propagationleftsideeven();}
                    }
            }
    }

```

```

        if (x==xmax-3) {propagationrightsideeven();}
        if ((x>3) && (x<xmax-3)) {propagationeven();}
    }
}

/* Code fragment 7: Recording of a new sytem's state */

for (x=1; x<xmax+1; x++)
{
    for (y=1; y<ymax+1; y++)
    {
        m[x][y]=nm[x][y]; vx[x][y]=nvx[x][y]; vy[x][y]=nvy[x][y];
        putpixel (x, y, m[x][y]*print);
    }
}

/* Code fragment 8: Data arrays resetting */

for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y++)
    {
        if (nm[x][y]<7)
            {nm[x][y]=0; nvx[x][y]=0; nvy[x][y]=0;
             i1[x][y]=0; i2[x][y]=0; i3[x][y]=0;
             i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;}
        else {nm[x][y]=7;}
    }
}

/* Code fragment 9: Printout macro */

setfillstyle(1,0);
bar(getmaxx()-90,getmaxy()-90,getmaxx(),getmaxy()-70);
outtextxy(getmaxx()-120,getmaxy()-80,"cycle");
outtextxy(getmaxx()-70,getmaxy()-80,gcvt(series,sig,str));

//Code fragment 9-A: Output FLOW.CPP
node=0; mass=0; velocity=0;
for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {mass=mass+m[x][y]; node++;
         velocity=velocity+vx[x][y];
        }
    }
flow=velocity/float(mass);
fprintf(output0,"%8d %8d %8d %8d %8d %8d %8.5f\n", transfer14, transfer25,
transfer36, cycle, node, mass, flow);

//Code fragment 9-B: Velocity profile of the flow
if (cycle>5000) {profile();}
series=series+1;
//getch();

} //The end of the main cycle

//Opening the data file PROFILE.CPP. Output
if ((output1=fopen("C:\\Outputs\\Poiseuil\\Profile2.cpp","w"))==NULL)
{
    printf("output file error\n");
    exit(0);
}

```

```

    }
for (y=3; y<ymax-2; y++)
    {fprintf(output1, "%5i %3.5f\n", y, V[y]/float(step));}

/* Code fragment 10: Final operations */

getch();
closegraph();
fclose (output0);
fclose (output1);
return (0);
} //The end of the main part of the algorithm

/*-----*/

//SUBROUTINES

/*-----*/
//Collision phase
int collision(void)
{
int cannel=0;
int mas=0;
float velx=0;
float vely=0;

nav2:
velx=vx[x][y]; vely=vy[x][y]; mas=m[x][y];
i1[x][y]=0; i2[x][y]=0; i3[x][y]=0; i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;

nav1:
cannel=0;
cannel=random(6);

    if (cannel==0)
        {if (i1[x][y]==1) {goto nav1;}
        i1[x][y]=1; mas=mas-1;}
    if (cannel==1)
        {if (i2[x][y]==1) {goto nav1;}
        i2[x][y]=1; mas=mas-1;}
    if (cannel==2)
        {if (i3[x][y]==1) {goto nav1;}
        i3[x][y]=1; mas=mas-1;}
    if (cannel==3)
        {if (i4[x][y]==1) {goto nav1;}
        i4[x][y]=1; mas=mas-1;}
    if (cannel==4)
        {if (i5[x][y]==1) {goto nav1;}
        i5[x][y]=1; mas=mas-1;}
    if (cannel==5)
        {if (i6[x][y]==1) {goto nav1;}
        i6[x][y]=1; mas=mas-1;}

//change of mass and velocity in the cell - has to be zero
    if (mas!=0) {goto nav1;}
    velx=velx+(0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-0.5*i4[x][y]-i5[x][y]-
0.5*i6[x][y]);
    if (velx!=0) {goto nav2;}
    vely=vely+(sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y]);
    if (vely!=0) {goto nav2;}
return(0);

```

```

}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationodd(void)
{
if (i1[x][y]==1)
    {
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
    nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]-0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x-1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
    nvy[x-1][y+1]=nvy[x-1][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]+0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
    }
return(0);
}

```

```

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationeven(void)
{
if (i1[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]+0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x+1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
    nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]-0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
    nvy[x+1][y-1]=nvy[x+1][y-1]-sinangle;}
    }
return(0);
}

```

```

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationleftsideodd(void)
{
if (i1[x][y]==1)
    {
    if (nm[xmax-3][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[xmax-3][y-1]=nm[xmax-3][y-1]+1; nvx[xmax-3][y-1]=nvx[xmax-
3][y-1]-0.5;
    nvy[xmax-3][y-1]=nvy[xmax-3][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[xmax-3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[xmax-3][y]=nm[xmax-3][y]+1; nvx[xmax-3][y]=nvx[xmax-3][y]-1;
    nvy[xmax-3][y]=nvy[xmax-3][y]-0;}
    }

if (i3[x][y]==1)
    {
    if (nm[xmax-3][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[xmax-3][y+1]=nm[xmax-3][y+1]+1; nvx[xmax-3][y+1]=nvx[xmax-
3][y+1]-0.5;
    nvy[xmax-3][y+1]=nvy[xmax-3][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]+0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
    }
return(0);
}

```

```

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationrightsideoodd(void)
{
if (i1[x][y]==1)
    {
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
    nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]-0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x-1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
    nvy[x-1][y+1]=nvy[x-1][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[3][y]=nm[3][y]+1; nvx[3][y]=nvx[3][y]+1;
    nvy[3][y]=nvy[3][y]+0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
    }
return(0);
}

```

```

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationleftsideeven(void)
{
if (i1[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[xmax-3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[xmax-3][y]=nm[xmax-3][y]+1; nvx[xmax-3][y]=nvx[xmax-3][y]-1;
    nvy[xmax-3][y]=nvy[xmax-3][y]+0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x+1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
    nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]-0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
    nvy[x+1][y-1]=nvy[x+1][y-1]-sinangle;}
    }
return(0);
}

/*-----*/

```

```

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationrightsideeven(void)
{
if (i1[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvx[x][y]=nvx[x][y]-0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvx[x-1][y]=nvx[x-1][y]+0;}
    }

if (i3[x][y]==1)
    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvx[x][y+1]=nvx[x][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[3][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[3][y+1]=nm[3][y+1]+1; nvx[3][y+1]=nvx[3][y+1]+0.5;
    nvx[3][y+1]=nvx[3][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]+0;}
    else {nm[3][y]=nm[3][y]+1; nvx[3][y]=nvx[3][y]+1;
    nvx[3][y]=nvx[3][y]-0;}
    }

if (i6[x][y]==1)
    {
    if (nm[3][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[3][y-1]=nm[3][y-1]+1; nvx[3][y-1]=nvx[3][y-1]+0.5;
    nvx[3][y-1]=nvx[3][y-1]-sinangle;}
    }
return(0);
}

/*-----*/

```

```
//Pressure gradient
float turnright(void)
{
    if ((i1[x][y]==1)&&(i4[x][y]==0))
        {i1[x][y]=0; i4[x][y]=1; m[x][y]=m[x][y];
        transfer14++;}

    if ((i2[x][y]==1)&&(i5[x][y]==0))
        {i2[x][y]=0; i5[x][y]=1; m[x][y]=m[x][y];
        transfer25=transfer25+2;}

    if ((i3[x][y]==1)&&(i6[x][y]==0))
        {i3[x][y]=0; i6[x][y]=1; m[x][y]=m[x][y];
        transfer36++;}
return(0);
}

/*-----*/

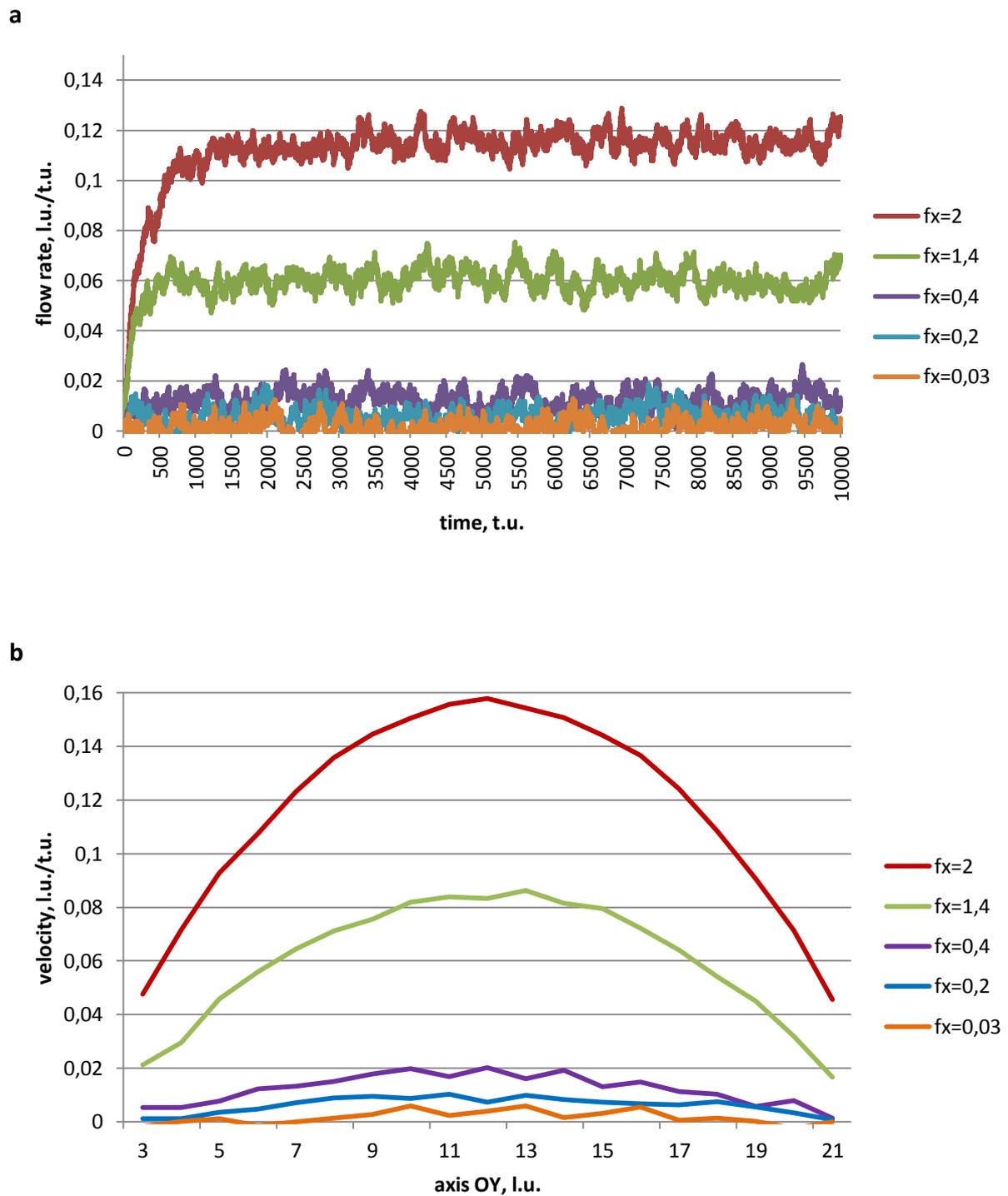
//Velocity profile
float profile(void)
{
float velocity;
int particles;
    for (y=3; y<ymax-2; y++)
        { velocity=0; particles=0;
        for (x=7; x<xmax-2; x++)
            {velocity=velocity+vx[x][y];
            particles=particles+m[x][y];}
        V[y]=V[y]+velocity/float(particles);
        }
    step++;
return (0);
}

/*-----*/
```

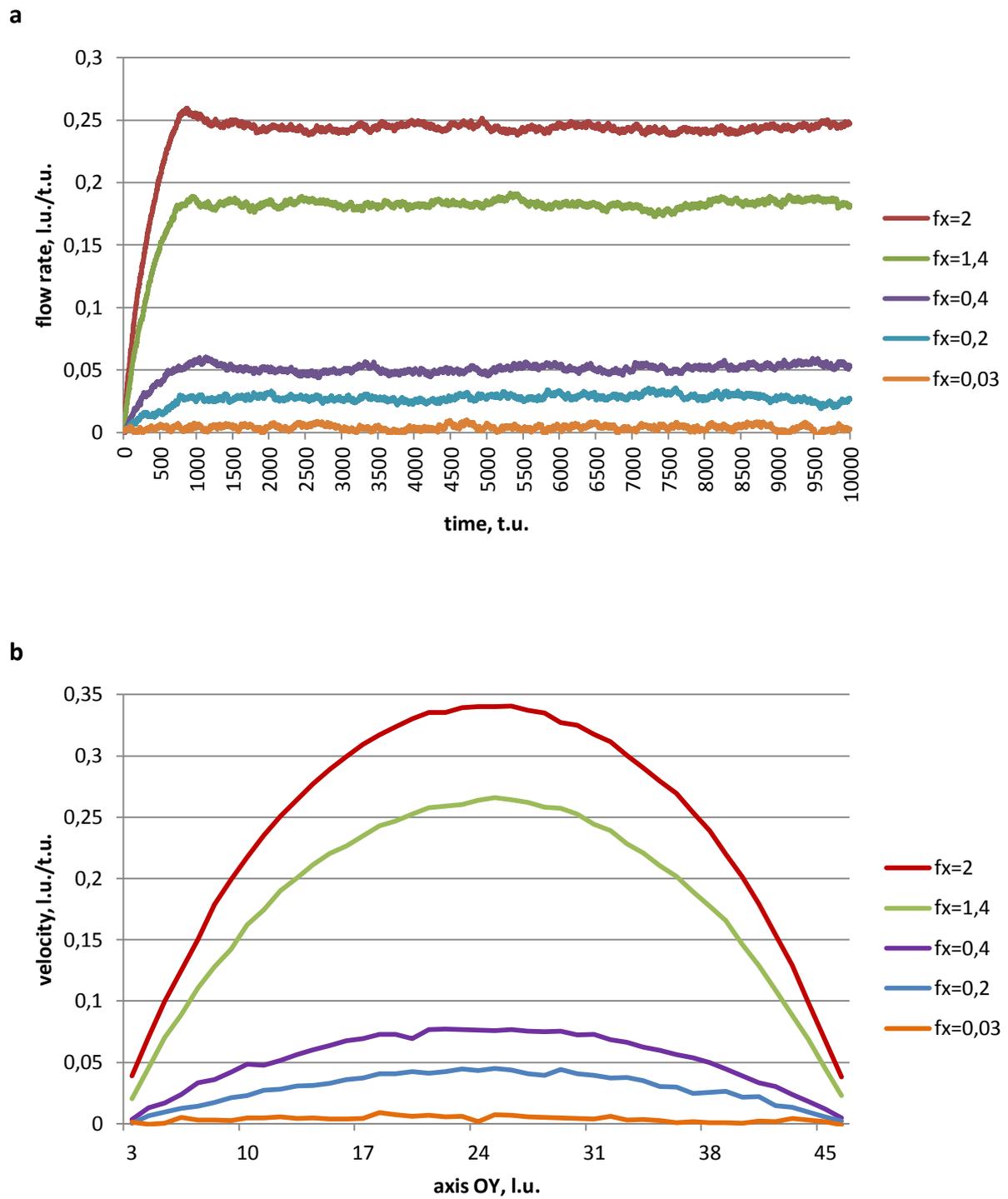
**APPENDIX J**

Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Flow rate as a function of time and velocity profiles for various width of the channel  $d$

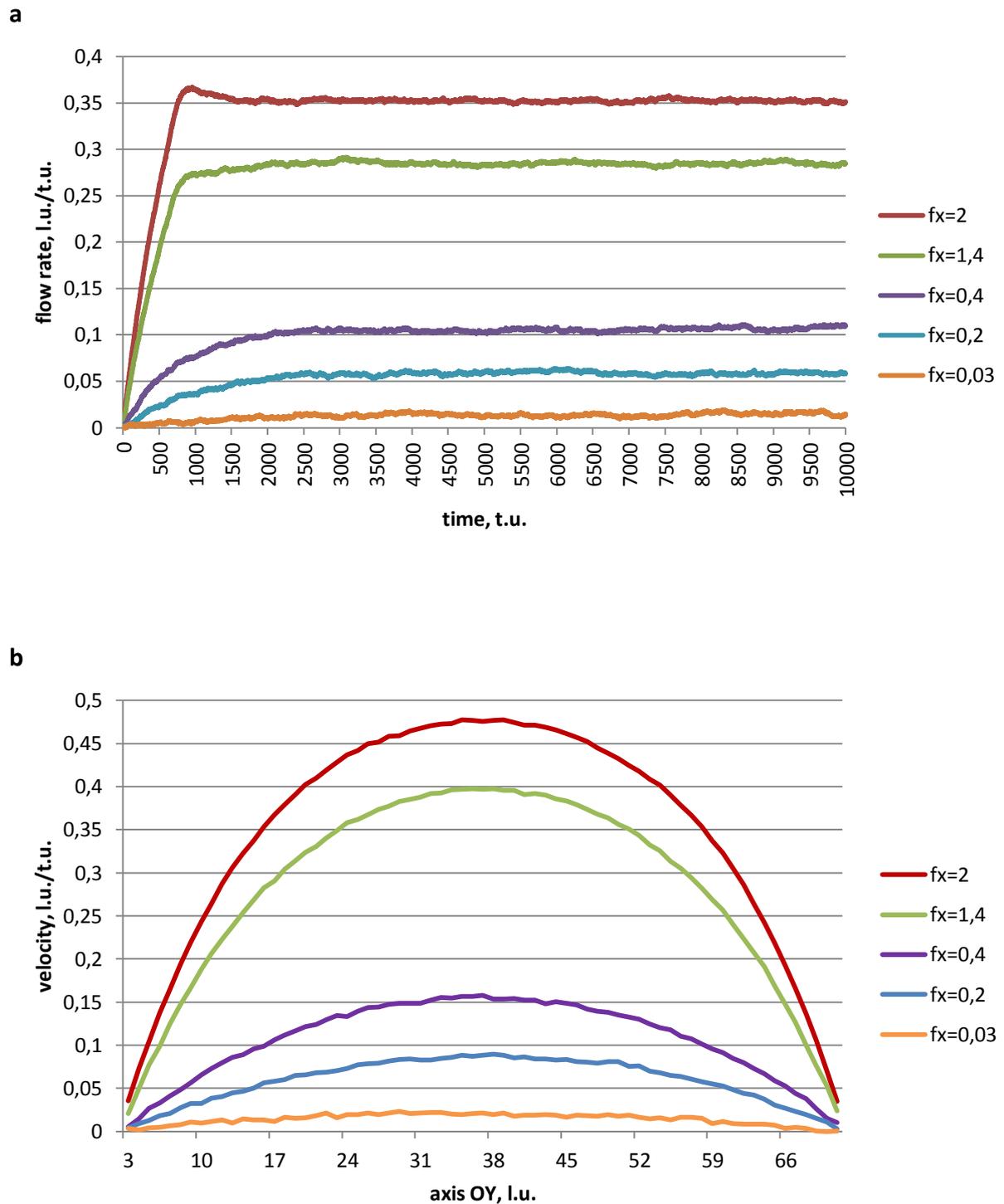
---



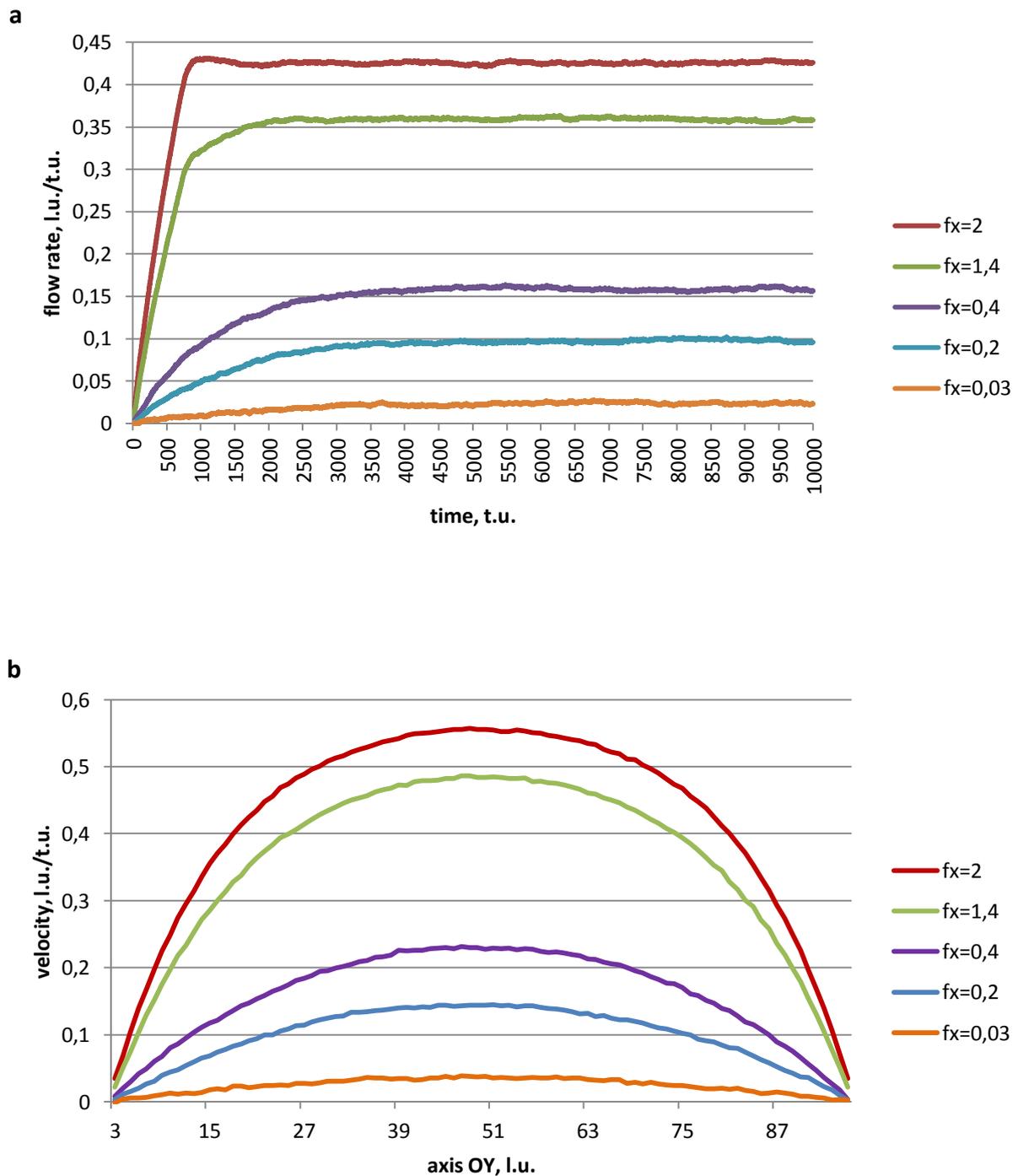
**Figure J-1:** Computer simulation of the Poiseuille flow: **a** – the flow rate as a function of time for the channel of the size  $L = 550$  l.u.,  $d = 25\sqrt{3}/2$  l.u. and various  $f_x$ ; **b** - the velocity profile of the flow presented by values of the  $x$  component of flow velocity averaged over the whole channel length in a steady state region of the flow



**Figure J-2:** Computer simulation of the Poiseuille flow: **a** – the flow rate as a function of time for the channel of the size  $L = 550$  l.u.,  $d = 50\sqrt{3}/2$  l.u. and various  $f_x$ ; **b** - the velocity profile of the flow presented by values of the  $x$  component of flow velocity averaged over the whole channel length in a steady state region of the flow



**Figure J-3:** Computer simulation of the Poiseuille flow: **a** – the flow rate as a function of time for the channel of the size  $L = 550$  l.u.,  $d = 75\sqrt{3}/2$  l.u. and various  $f_x$ ; **b** - the velocity profile of the flow presented by values of the  $x$  component of flow velocity averaged over the whole channel length in a steady state region of the flow

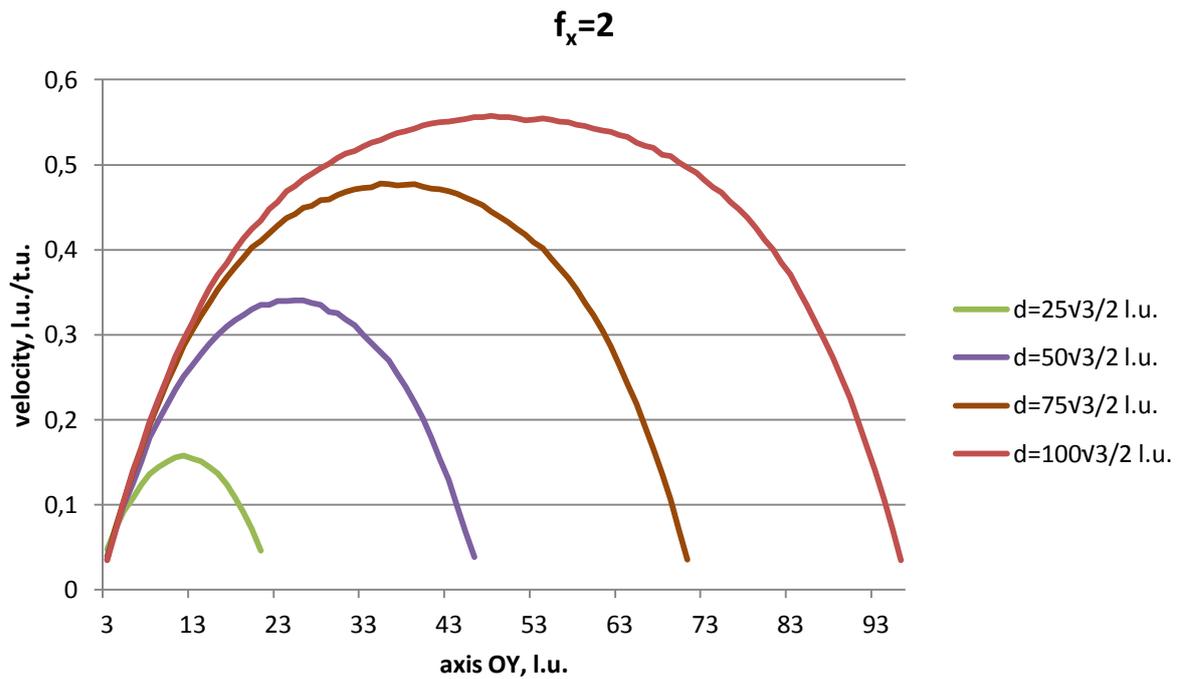


**Figure J-4:** Computer simulation of the Poiseuille flow: **a** – the flow rate as a function of time for the channel of the size  $L = 550$  l. u.,  $d = 100\sqrt{3}/2$  l. u. and various  $f_x$ ; **b** - the velocity profile of the flow presented by values of the  $x$  component of flow velocity averaged over the whole channel length in a steady state region of the flow

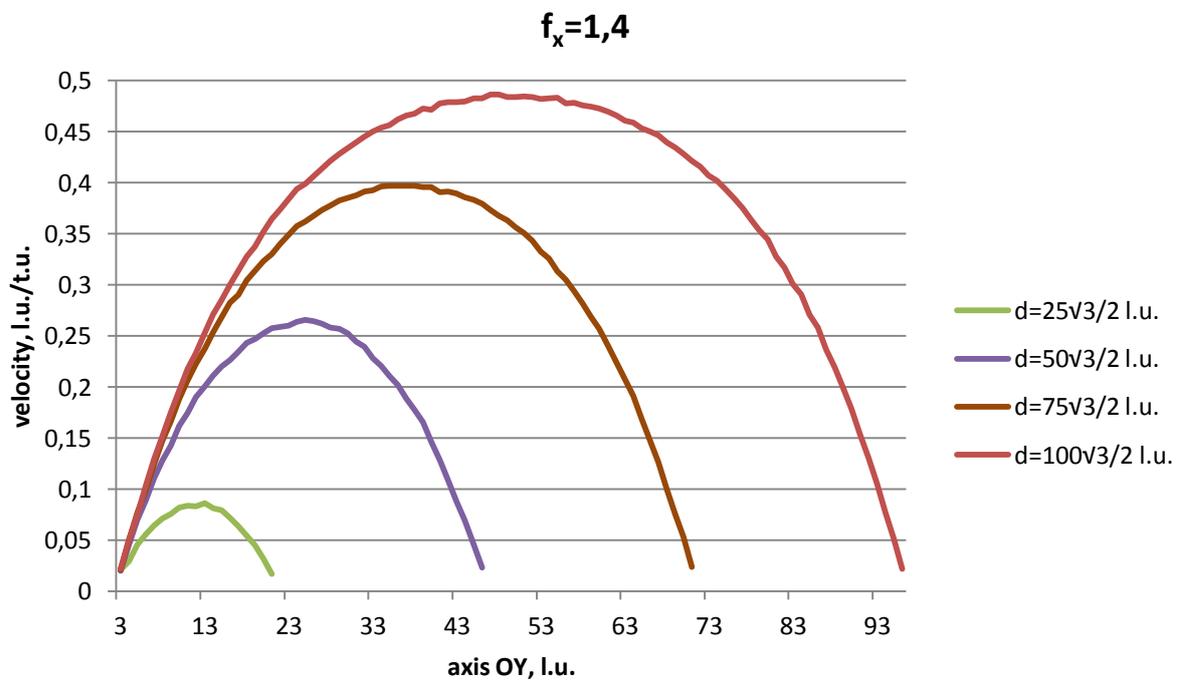
**APPENDIX K**

Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Parabolic velocity profiles of the flow for various width of the channel  $d$  and for different pressure gradient created by various values of the parameter  $f_x$

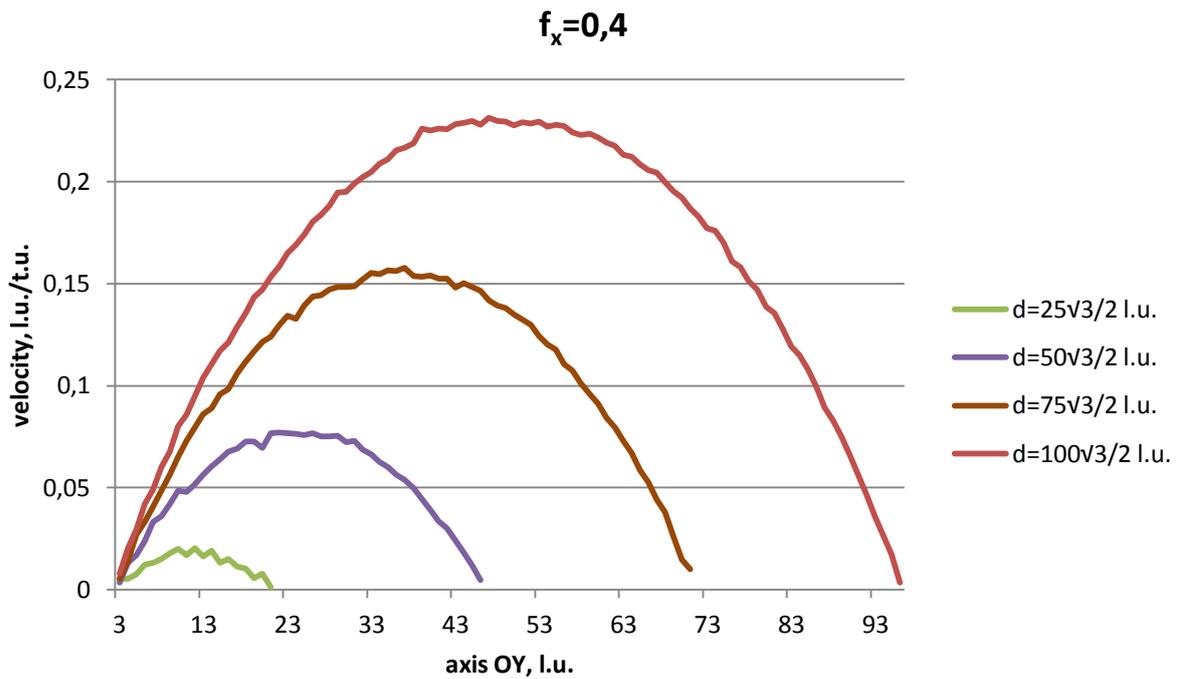
---



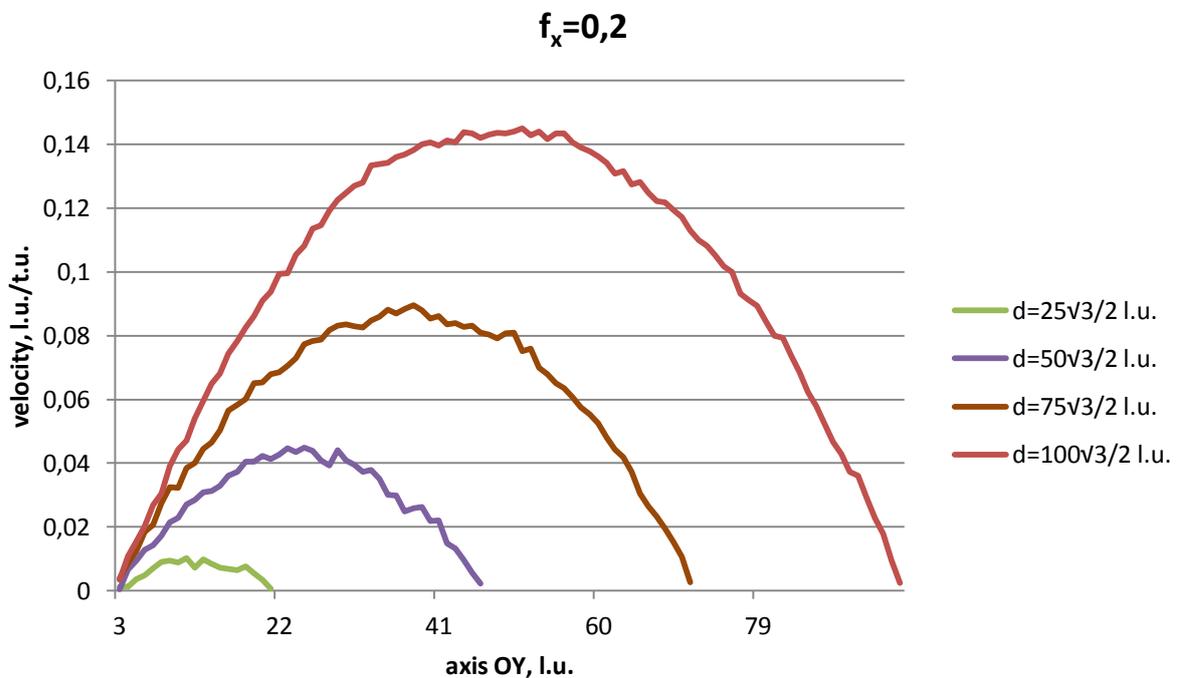
**Figure K-1:** Parabolic velocity profiles for various width of the channel  $d$  and for pressure gradient created by  $f_x = 2$ . The length of the channel is 550 l. u.



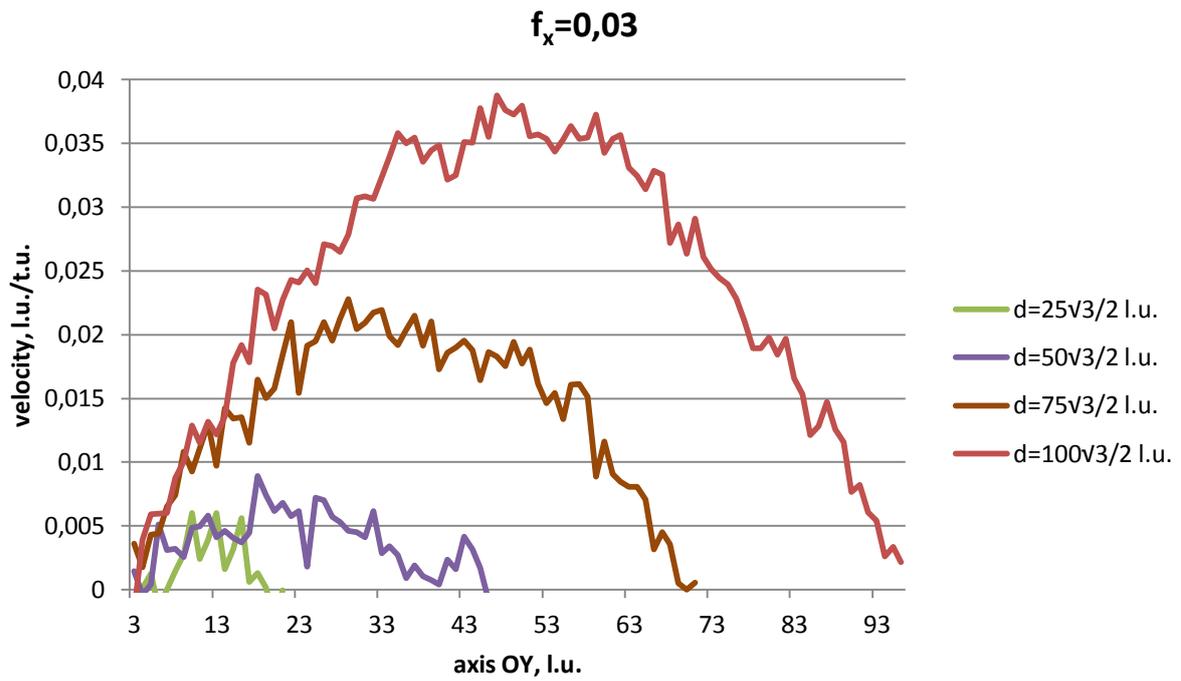
**Figure K-2:** Parabolic velocity profiles for various width of the channel  $d$  and for pressure gradient created by  $f_x = 1,4$ . The length of the channel is 550 l. u.



**Figure K-3:** Parabolic velocity profiles for various width of the channel  $d$  and for pressure gradient created by  $f_x = 0,4$ . The length of the channel is 550 l. u.



**Figure K-4:** Parabolic velocity profiles for various width of the channel  $d$  and for pressure gradient created by  $f_x = 0,2$ . The length of the channel is 550 l. u.

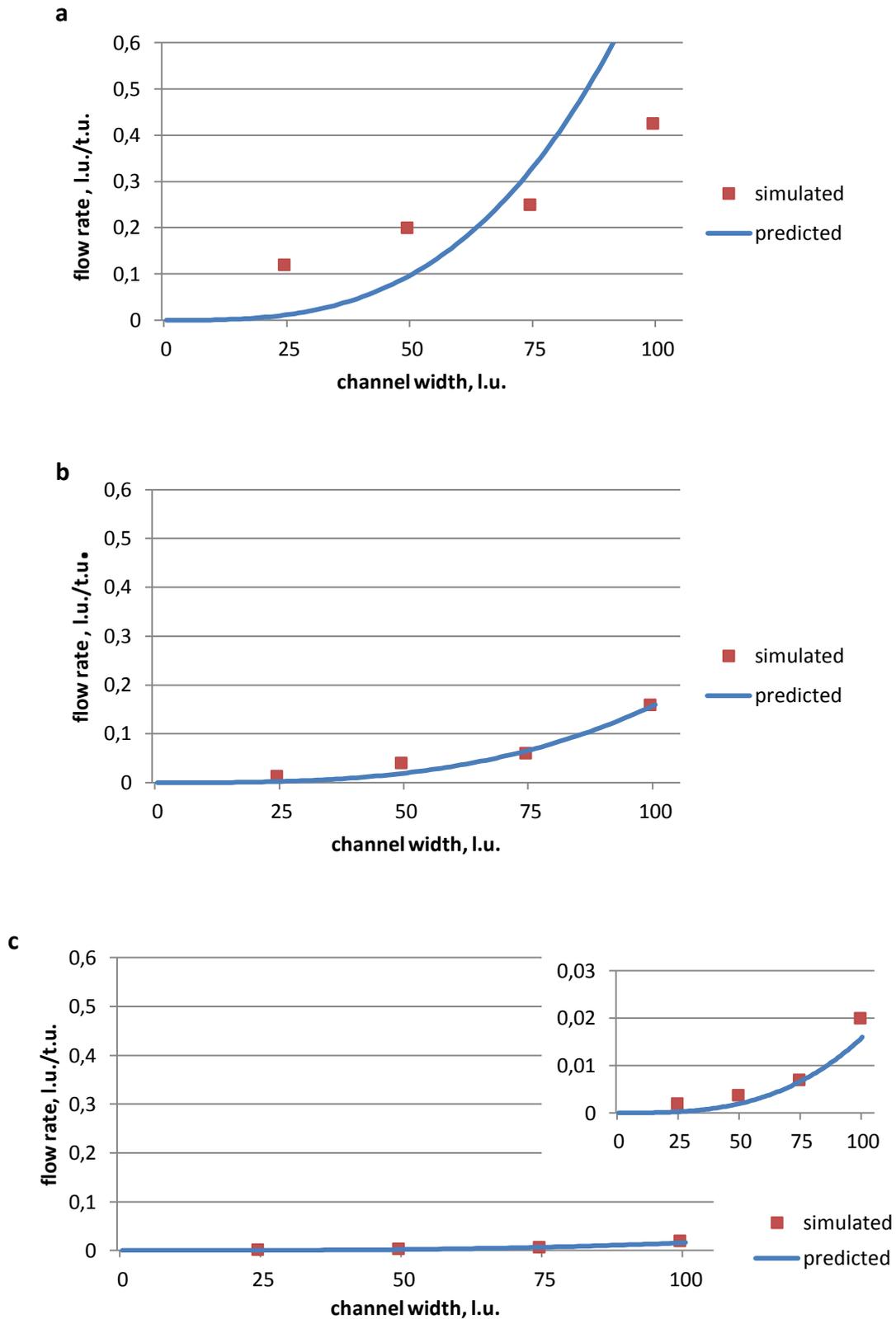


**Figure K-5:** Parabolic velocity profiles for various width of the channel  $d$  and for pressure gradient created by  $f_x = 0,03$ . The length of the channel is 550 l. u.

**APPENDIX L**

Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Flow rate as a function of channel width  $d$  for a pressure gradient created by various  $f_x$

---

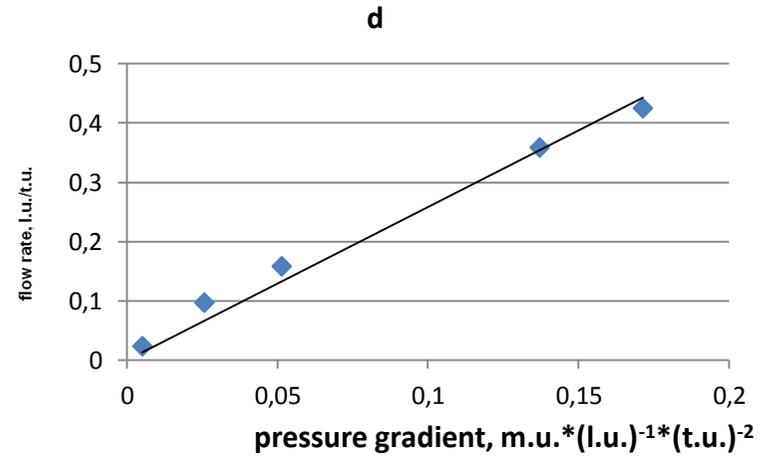
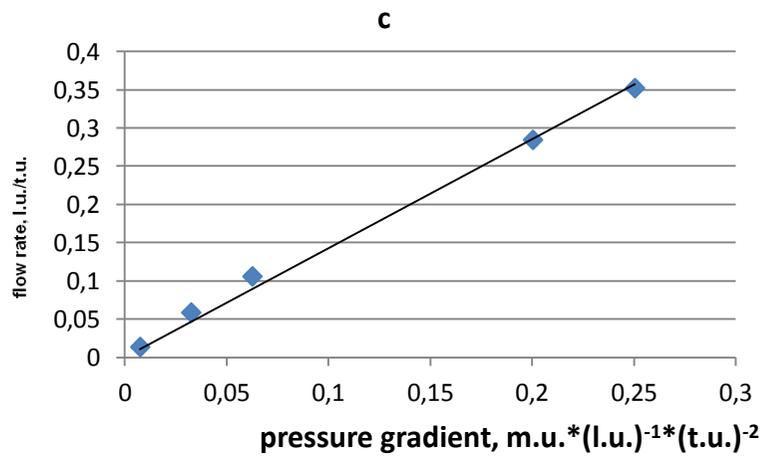
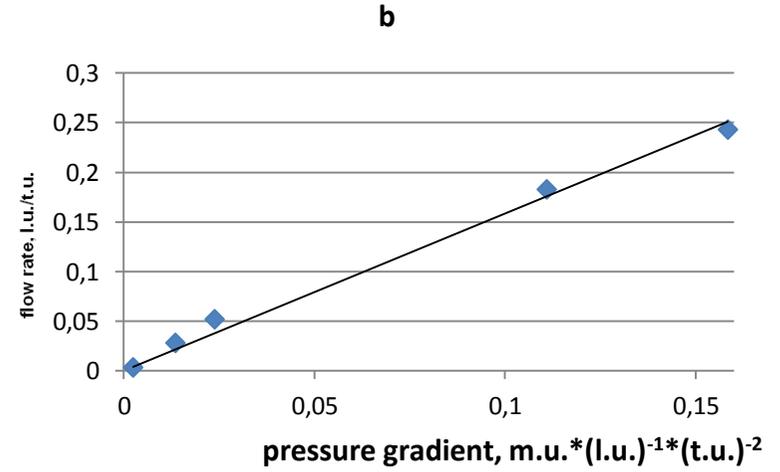
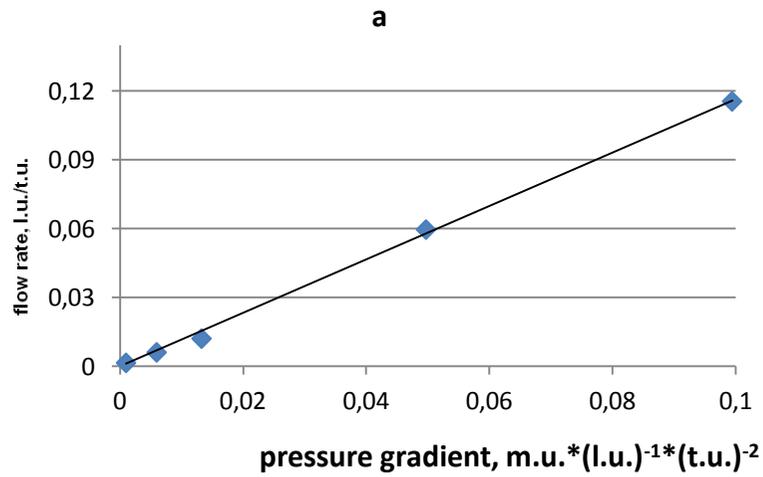


**Figure L-1:** Predicted and simulated flow rate as a function of channel width for a pressure gradient created by  $f_x = 2$  (a),  $f_x = 0,4$  (b) and  $f_x = 0,03$  (c). The range of the channel width is  $d = 25 \div 100$  l.u. The length of the channel  $L = 550$  l.u.

**APPENDIX M**

Verification of the FHP-1 Lattice Gas Cellular Automata algorithm for Poiseuille flow. Validation of the Darcy's law for various width of the channel  $d$ .

---



**Figure M-1:** Verification of the Darcy's law. The flow rate as a function of the pressure gradient for various channel width:  $d = 25 \sqrt{3}/2$  l.u. (a),  $d = 50 \sqrt{3}/2$  l.u.; (b),  $d = 75 \sqrt{3}/2$  l.u. (c) and  $d = 100 \sqrt{3}/2$  l.u. (d)

**APPENDIX N**

The FHP-1 Lattice Gas Cellular Automata algorithm for simulation of the fluid flow through porous media.

---

```

/FHP-1 LGCA for fluid flow through porous media simulation

/*Code fragment 1: Header files and initialization of the simulation box */

//Definition of standard library functions
# include <graphics.h>
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>
# include <float.h>
# include <time.h>
# define DIRX 450
# define DIRY 250

//Declaration of variables
int x, y, xmax=449, ymax=249;
float vx[DIRX][DIRY], nvx[DIRX][DIRY];
float vy[DIRX][DIRY], nvy[DIRX][DIRY];
int m[DIRX][DIRY], nm[DIRX][DIRY];
int i1[DIRX][DIRY], i2[DIRX][DIRY], i3[DIRX][DIRY], i4[DIRX][DIRY],
i5[DIRX][DIRY], i6[DIRX][DIRY];
int mass=0, node=0;
float velfieldx[DIRX][DIRY], velfielddy[DIRX][DIRY], velocity=0;
int ventilator=5, force=1;
float sinangle=0.866025403, step=0, pi=3.14, alfa, b1, b2;
float flow, V[DIRY];
int pco=2;
int porousmedium=5, i=45, angle=35;
int pore=0, fibre=0;
int transfer14=0, transfer25=0, transfer36=0;
int fluid=3, obstacle=4, hole=0;
int sig=15;
char str[25];
int I1, I2, I3, I4, I5, I6;
int cycle, cmax=10000, series;

//Declaration of subroutines and output files
int collision(void);
float propagationodd(void);
float propagationeven(void);
float propagationleftsideodd(void);
float propagationleftsideeven(void);
float propagationrightsideodd(void);
float propagationrightsideeven(void);
float turnright(void);
float profile(void);
float velocityfield(void);
FILE *output1;
FILE *output2;
FILE *output3;
FILE *output4;
FILE *output5;
FILE *output6;

/*-----*/
/* Beginning of a main part of the program */
int main()
{

/* Code fragment 2: Graphic outputs setting */

```

```

int gdriver = DETECT, gmode, errorcode;

//initialize graphics and local variabls
initgraph (&gdriver, &gmode, "c:\\TC\\BGI");

//read result of initialization
errorcode = graphresult();

//an error occurred
if (errorcode != grOk)
{
    printf ("Graphics error: %s\n", grapherrormsg(errorcode));
    printf ("Press any key to halt:");
    getch();
    exit(1);
}

/* Code fragment 3: Creation of the simulation domain and initial state of
the simulated system */

//Data arrays resetting
for (x=0; x<xmax+1; x++)
{
    for (y=0; y<ymax+1; y++)
    {
        m[x][y]=0;
        nm[x][y]=0;
        vx[x][y]=0;
        nvx[x][y]=0;
        vy[x][y]=0;
        nvy[x][y]=0;
    }
}

//Creation of solid boundaries of the simulation box
for (x=1; x<xmax; x++)
{
    m[x][1]=7;
    m[x][2]=7;
    m[x][ymax-1]=7;
    m[x][ymax-2]=7;

    nm[x][1]=7;
    nm[x][2]=7;
    nm[x][ymax-1]=7;
    nm[x][ymax-2]=7;
    putpixel (x, 1, m[x][1]);
    putpixel (x, ymax-1, m[x][ymax-1]);
}

//randomize();

//porous medium
for (x=3; x<xmax-2; x++)
{
    for (y=3; y<ymax-2; y++)
    {
        if (random(101)<porousmedium)
            {m[x][y]=7; nm[x][y]=7;}
    }
}

//declined porous medium
alfa=angle*pi/180;

```

```

b1=cos(alfa)+0;
b2=cos(alfa)+b1+2*i;

for (x=1; x<xmax; x++)
{for (y=3; y<ymax-2; y++)
  {
    if (y>(x*cos(alfa)-b1)) {m[x][y]=0; nm[x][y]=0;}
  }
}
for (x=1; x<xmax; x++)
{for (y=3; y<ymax-2; y++)
  {
    if (y<(x*cos(alfa)-b2)) {m[x][y]=0; nm[x][y]=0;}
  }
}

//porosity calculation
for (x=1; x<xmax; x++)
  {for (y=3; y<ymax-2; y++)
    {
      if ((y>(x*cos(alfa)-b2)) && (y<(x*cos(alfa)-b1)))
        {if (m[x][y]!=7) {pore++;} else {fibre++;}}
    }
  }

/* Code fragment 4: Occupation of cannels by fluid moving particles */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
  {
    for (y=3; y<ymax-2; y=y+2)
      {
        if (m[x][y]!=7)
          {
            m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

            if (m[x-1][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

            if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}

            if (m[x-1][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

            if (m[x][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

            if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

            if (m[x][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

            //the total particles velocity in the node
            vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
            vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
          }
      }
  }
}

```

```

//even rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=4; y<ymax-2; y=y+2)
            {
                if (m[x][y]!=7)
                    {
                        m[x][y]=0; vx[x][y]=0; vy[x][y]=0;

                        if (m[x][y-1]<7) {I1=random(pco);}
if (I1==1) {m[x][y]=m[x][y]+1; i1[x][y]=1;} else {i1[x][y]=0;}

                        if (m[x-1][y]<7) {I2=random(pco);}
if (I2==1) {m[x][y]=m[x][y]+1; i2[x][y]=1;} else {i2[x][y]=0;}

                        if (m[x][y+1]<7) {I3=random(pco);}
if (I3==1) {m[x][y]=m[x][y]+1; i3[x][y]=1;} else {i3[x][y]=0;}

                        if (m[x+1][y+1]<7) {I4=random(pco);}
if (I4==1) {m[x][y]=m[x][y]+1; i4[x][y]=1;} else {i4[x][y]=0;}

                        if (m[x+1][y]<7) {I5=random(pco);}
if (I5==1) {m[x][y]=m[x][y]+1; i5[x][y]=1;} else {i5[x][y]=0;}

                        if (m[x+1][y-1]<7) {I6=random(pco);}
if (I6==1) {m[x][y]=m[x][y]+1; i6[x][y]=1;} else {i6[x][y]=0;}

                        // the total particles velocity in the node
                        vx[x][y]=vx[x][y]+0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-
0.5*i4[x][y]-i5[x][y]-0.5*i6[x][y];
                        vy[x][y]=vy[x][y]+sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y];
                    }
            }
}

/* Code fragment 5: Graphical outputs of the initial system configuration
*/

for (x=1; x<xmax+1; x++)
    {
        for (y=1; y<ymax+1; y++)
            {
                putpixel (x, y, m[x][y]);
            }
    }

//Opening the data file FLOW.CPP
if ((output1=fopen("C:\\Outputs\\Filter\\Flow01.cpp", "w"))==NULL)
    {
        printf("output file error\n");
        exit(0);
    }
//Opening the data file INFO.CPP
if ((output2=fopen("C:\\Outputs\\Filter\\Info01.cpp", "w"))==NULL)
    {
        printf("output file error\n");
        exit(0);
    }
/*-----*/

/* Code fragment 6: The main cycle of the algorithm */

```

```

for (cycle=0; cycle<cmax+1; cycle++)
{
/* Code fragment 6-A: Collision phase */

for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if ((m[x][y]>0)&&(m[x][y]!=7)) {collision();}
        }
    }

/* Code fragment 6-B: Pressure gradient */

transfer14=0; transfer25=0; transfer36=0;
for (x=3; x<ventilator; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if ((m[x][y]>0)&&(m[x][y]!=7))
                { if (random(force)<100) {turnright();}}
        }
    }

//Porous medium
for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {
            if (m[x][y]==7) {nm[x][y]=7;}
        }
    }

/* Code fragment 6-C: Propagation phase */

//odd rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=3; y<ymax-2; y=y+2)
            {
                if ((m[x][y]>0)&&(m[x][y]!=7))
                    {
                        if (x==3) {propagationleftsideodd();}
                        if (x==xmax-3) {propagationrightsideodd();}
                        if ((x>3)&&(x<xmax-3)) {propagationodd();}
                    }
            }
    }

//even rows of the lattice
for (x=3; x<xmax-2; x++)
    {
        for (y=4; y<ymax-2; y=y+2)
            {
                if ((m[x][y]>0)&&(m[x][y]!=7))
                    {
                        if (x==3) {propagationleftsideeven();}
                        if (x==xmax-3) {propagationrightsideeven();}
                        if ((x>3)&&(x<xmax-3)) {propagationeven();}
                    }
            }
    }

/* Code fragment 7: Recording of a new sytem's state */

```

```

for (x=1; x<xmax+1; x++)
    {
    for (y=1; y<ymax+1; y++)
        {
        m[x][y]=nm[x][y]; vx[x][y]=nvx[x][y]; vy[x][y]=nvy[x][y];
        if ((m[x][y]>0)&&(m[x][y]<7)) {putpixel(x, y, fluid);}
        if (m[x][y]==7) {putpixel(x, y, obstacle);}
        if (m[x][y]==0) {putpixel(x, y, hole);}
        }
    }

/* Code fragment 8: Data arrays resetting */

for (x=3; x<xmax-2; x++)
    {
    for (y=3; y<ymax-2; y++)
        {
        if (nm[x][y]<7)
            {nm[x][y]=0; nvx[x][y]=0; nvy[x][y]=0;
            i1[x][y]=0; i2[x][y]=0; i3[x][y]=0;
            i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;}
        else {nm[x][y]=7;}
        }
    }

/* Code fragment 9: Printout macro */

setfillstyle(1,0);
bar(getmaxx()-90,getmaxy()-90,getmaxx(),getmaxy()-70);
outtextxy(getmaxx()-120,getmaxy()-80,"cycle");
outtextxy(getmaxx()-70,getmaxy()-80,gcvt(series,sig,str));

//Code fragment 9-A: Output FLOW.CPP
node=0; mass=0; velocity=0;
for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {mass=mass+m[x][y]; node++;
        velocity=velocity+vx[x][y];
        }
    }
flow=velocity/float(mass);
fprintf(output1,"%8d %8d %8d %8d\n", transfer14, transfer25, transfer36,
node);
fprintf(output2,"%8d %8.5f %8d\n", cycle, flow, mass);

//Code fragment 9-B: Distribution of velocity vectors of moving particles
if (cycle>2000)
    {profile();
    velocityfield();}

series=series+1;
//getch();

} //The end of the main cycle

/*-----*/

//Opening the data file PROFILE.CPP. Output
if ((output3=fopen("C:\\Outputs\\Filter\\Profile1.cpp","w"))==NULL)
    {
    printf("output file error\n");

```

```

        exit(0);
    }
for (y=3; y<ymax-2; y++)
    {fprintf(output3, "%5i %3.5f\n", y, V[y]/float(step));}

//Opening the data file POROUS.CPP. Output
if ((output4=fopen("C:\\Outputs\\Filter\\Porous1.cpp", "w"))==NULL)
    {
    printf("output file error\n");
    exit(0);
    }
        fprintf(output4, "%5i %5i\n", fibre, pore);

//Opening the data file VELFIEL.CPP. Output
if ((output5=fopen("C:\\Outputs\\Filter\\Velfiel1.cpp", "w"))==NULL)
    {
    printf("output file error\n");
    exit(0);
    }
for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {fprintf(output5, "%5i %5i %5.5f %5.5f\n", x, y,
velfieldx[x][y]/float(step), velfiel dy[x][y]/float(step));}
    }

//Opening the data file FIGURE.CPP. Output
if ((output6=fopen("C:\\Outputs\\Filter\\Figure1.cpp", "w"))==NULL)
    {
    printf("output file error\n");
    exit(0);
    }
for (x=3; x<xmax-2; x++)
    {for (y=3; y<ymax-2; y++)
        {fprintf(output6, "%5i %5i %5i\n", x, y, m[x][y]);}
    }

/*-----*/

/* Code fragment 10: Final operations */

getch();
closegraph();
fclose (output1);
fclose (output2);
fclose (output3);
fclose (output4);
fclose (output5);
fclose (output6);
return (0);
} //The end of the main part of the algorithm

/*-----*/

//SUBROUTINES

/*-----*/
//Collision phase
int collision(void)
{
int cannel=0;
int mas=0;
float velx=0;

```

```

float vely=0;

nav2:
velx=vx[x][y]; vely=vy[x][y]; mas=m[x][y];
i1[x][y]=0; i2[x][y]=0; i3[x][y]=0; i4[x][y]=0; i5[x][y]=0; i6[x][y]=0;

nav1:
cannel=0;
cannel=random(6);

    if (cannel==0)
        {if (i1[x][y]==1) {goto nav1;}
        i1[x][y]=1; mas=mas-1;}
    if (cannel==1)
        {if (i2[x][y]==1) {goto nav1;}
        i2[x][y]=1; mas=mas-1;}
    if (cannel==2)
        {if (i3[x][y]==1) {goto nav1;}
        i3[x][y]=1; mas=mas-1;}
    if (cannel==3)
        {if (i4[x][y]==1) {goto nav1;}
        i4[x][y]=1; mas=mas-1;}
    if (cannel==4)
        {if (i5[x][y]==1) {goto nav1;}
        i5[x][y]=1; mas=mas-1;}
    if (cannel==5)
        {if (i6[x][y]==1) {goto nav1;}
        i6[x][y]=1; mas=mas-1;}

//change of mass and velocity in the cell - has to be zero
    if (mas!=0) {goto nav1;}
    velx=velx+(0.5*i1[x][y]+i2[x][y]+0.5*i3[x][y]-0.5*i4[x][y]-i5[x][y]-
0.5*i6[x][y]);
    if (velx!=0) {goto nav2;}
    vely=vely+(sinangle*i1[x][y]-sinangle*i3[x][y]-
sinangle*i4[x][y]+sinangle*i6[x][y]);
    if (vely!=0) {goto nav2;}
return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationodd(void)
{
if (i1[x][y]==1)
    {
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
    nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvy[x-1][y]=nvy[x-1][y]-0;}
    }
}

```

```

if (i3[x][y]==1)
{
    if (nm[x-1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
    nvx[x-1][y+1]=nvx[x-1][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvx[x][y+1]=nvx[x][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]-0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvx[x+1][y]=nvx[x+1][y]+0;}
}

if (i6[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationeven(void)
{
if (i1[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvx[x][y]=nvx[x][y]-0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvx[x-1][y]=nvx[x-1][y]+0;}
}
}

```

```

if (i3[x][y]==1)
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[x+1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
    nvy[x+1][y+1]=nvy[x+1][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvy[x+1][y]=nvy[x+1][y]-0;}
}

if (i6[x][y]==1)
{
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
    nvy[x+1][y-1]=nvy[x+1][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationleftsideodd(void)
{
if (i1[x][y]==1)
{
    if (nm[xmax-3][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[xmax-3][y-1]=nm[xmax-3][y-1]+1; nvx[xmax-3][y-1]=nvx[xmax-
3][y-1]-0.5;
    nvy[xmax-3][y-1]=nvy[xmax-3][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
    if (nm[xmax-3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[xmax-3][y]=nm[xmax-3][y]+1; nvx[xmax-3][y]=nvx[xmax-3][y]-1;
    nvy[xmax-3][y]=nvy[xmax-3][y]-0;}
}
}

```

```

if (i3[x][y]==1)
{
    if (nm[xmax-3][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[xmax-3][y+1]=nm[xmax-3][y+1]+1; nvx[xmax-3][y+1]=nvx[xmax-
3][y+1]-0.5;
        nvy[xmax-3][y+1]=nvy[xmax-3][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
        nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
        nvy[x+1][y]=nvy[x+1][y]+0;}
}

if (i6[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
        nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in odd rows of the lattice (bounce-back reflection)
float propagationrightsideodd(void)
{
if (i1[x][y]==1)
{
    if (nm[x-1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x-1][y-1]=nm[x-1][y-1]+1; nvx[x-1][y-1]=nvx[x-1][y-1]-0.5;
        nvy[x-1][y-1]=nvy[x-1][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]+0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
        nvy[x-1][y]=nvy[x-1][y]-0;}
}
}

```

```

if (i3[x][y]==1)
{
    if (nm[x-1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x-1][y+1]=nm[x-1][y+1]+1; nvx[x-1][y+1]=nvx[x-1][y+1]-0.5;
    nvy[x-1][y+1]=nvy[x-1][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]+0.5;
    nvy[x][y+1]=nvy[x][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
    if (nm[3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[3][y]=nm[3][y]+1; nvx[3][y]=nvx[3][y]+1;
    nvy[3][y]=nvy[3][y]+0;}
}

if (i6[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]+0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationleftsideeven(void)
{
if (i1[x][y]==1)
{
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvy[x][y]=nvy[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvy[x][y-1]=nvy[x][y-1]-sinangle;}
}

if (i2[x][y]==1)
{
    if (nm[xmax-3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvy[x][y]=nvy[x][y]-0;}
    else {nm[xmax-3][y]=nm[xmax-3][y]+1; nvx[xmax-3][y]=nvx[xmax-3][y]-1;
    nvy[xmax-3][y]=nvy[xmax-3][y]+0;}
}

if (i3[x][y]==1)

```

```

    {
    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvx[x][y+1]=nvx[x][y+1]+sinangle;}
    }

if (i4[x][y]==1)
    {
    if (nm[x+1][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x+1][y+1]=nm[x+1][y+1]+1; nvx[x+1][y+1]=nvx[x+1][y+1]+0.5;
    nvx[x+1][y+1]=nvx[x+1][y+1]+sinangle;}
    }

if (i5[x][y]==1)
    {
    if (nm[x+1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]+0;}
    else {nm[x+1][y]=nm[x+1][y]+1; nvx[x+1][y]=nvx[x+1][y]+1;
    nvx[x+1][y]=nvx[x+1][y]-0;}
    }

if (i6[x][y]==1)
    {
    if (nm[x+1][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x+1][y-1]=nm[x+1][y-1]+1; nvx[x+1][y-1]=nvx[x+1][y-1]+0.5;
    nvx[x+1][y-1]=nvx[x+1][y-1]-sinangle;}
    }
return(0);
}

/*-----*/

//Propagation phase in even rows of the lattice (bounce-back reflection)
float propagationrightsideeven(void)
{
if (i1[x][y]==1)
    {
    if (nm[x][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[x][y-1]=nm[x][y-1]+1; nvx[x][y-1]=nvx[x][y-1]-0.5;
    nvx[x][y-1]=nvx[x][y-1]-sinangle;}
    }

if (i2[x][y]==1)
    {
    if (nm[x-1][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+1;
        nvx[x][y]=nvx[x][y]-0;}
    else {nm[x-1][y]=nm[x-1][y]+1; nvx[x-1][y]=nvx[x-1][y]-1;
    nvx[x-1][y]=nvx[x-1][y]+0;}
    }

if (i3[x][y]==1)
    {

```

```

    if (nm[x][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]+0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[x][y+1]=nm[x][y+1]+1; nvx[x][y+1]=nvx[x][y+1]-0.5;
    nvx[x][y+1]=nvx[x][y+1]+sinangle;}
}

if (i4[x][y]==1)
{
    if (nm[3][y+1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]-sinangle;}
    else {nm[3][y+1]=nm[3][y+1]+1; nvx[3][y+1]=nvx[3][y+1]+0.5;
    nvx[3][y+1]=nvx[3][y+1]+sinangle;}
}

if (i5[x][y]==1)
{
    if (nm[3][y]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-1;
        nvx[x][y]=nvx[x][y]+0;}
    else {nm[3][y]=nm[3][y]+1; nvx[3][y]=nvx[3][y]+1;
    nvx[3][y]=nvx[3][y]-0;}
}

if (i6[x][y]==1)
{
    if (nm[3][y-1]==7)
        {nm[x][y]=nm[x][y]+1; nvx[x][y]=nvx[x][y]-0.5;
        nvx[x][y]=nvx[x][y]+sinangle;}
    else {nm[3][y-1]=nm[3][y-1]+1; nvx[3][y-1]=nvx[3][y-1]+0.5;
    nvx[3][y-1]=nvx[3][y-1]-sinangle;}
}
return(0);
}

/*-----*/

//Pressure gradient
float turnright(void)
{
    if ((i1[x][y]==1)&&(i4[x][y]==0))
        {i1[x][y]=0; i4[x][y]=1; m[x][y]=m[x][y];
        transfer14++;}

    if ((i2[x][y]==1)&&(i5[x][y]==0))
        {i2[x][y]=0; i5[x][y]=1; m[x][y]=m[x][y];
        transfer25=transfer25+2;}

    if ((i3[x][y]==1)&&(i6[x][y]==0))
        {i3[x][y]=0; i6[x][y]=1; m[x][y]=m[x][y];
        transfer36++;}
return(0);
}

/*-----*/

//Velocity profile
float profile(void)
{
float velocity;
int particles;

```

```
for (y=3; y<ymax-2; y++)
{
    velocity=0; particles=0;
    for (x=7; x<xmax-2; x++)
    {
        if (m[x][y]!=7)
            {velocity=velocity+vx[x][y];
             particles=particles+m[x][y];}
    }
    V[y]=V[y]+velocity/float(particles);
}
step++;
return (0);
}

/*-----*/

//The field of velocities vectors
float velocityfield(void)
{
for (x=3; x<xmax-2; x++)
    {
    for (y=3; y<ymax-2; y++)
        {
        if (m[x][y]!=7)
            {velfieldx[x][y]=velfieldx[x][y]+vx[x][y];
             velfieldy[x][y]=velfieldy[x][y]+vy[x][y];}
        }
    }
return (0);
}

/*-----*/
```

**APPENDIX O**

The FHP-1 Lattice Gas Cellular Automata model for simulation of the fluid flow through porous media. Algorithms for data processing and their graphical representation.

---

```

/* FHP-1 LGCA for fluid flow in porous medium simulation */

/* X- and y- components of particles velocities - averaging in a space */

//Definition of standard library functions
# include <stdlib.h>
# include <stdio.h>
# include <conio.h>
# include <math.h>

//Declaration of variables
int x, y, xmax=0, ymax=0;
float velfieldx[450][250], velfielgy[450][250];
float xcomponent, ycomponent;
FILE *output, *output1;

/*-----*/
/* Beginning of a main part of the program */

int main(void)
{

//Opening the data file VELFIEL.CPP
if ((output=fopen("C:\\Outputs\\Filter\\Velfiel1.cpp", "r"))==NULL)
    {printf("input file error\n");
    exit(0);}

while(fscanf(output, "%i %i %f %f\n", &velfieldx, &velfielgy, &xcomponent,
&ycomponent) !=EOF)
    {
    velfieldx[x][y]=prx; velfielgy[x][y]=pry;
    if(xmax<x) {xmax=x;}
    if(ymax<y) {ymax=y;}
    }

//Opening the new data file VELFIEL.CPP. Averaging and saving outputs
if
((output1=fopen("C:\\Outputs\\Filter\\Graphic\\Velfiel1.cpp", "w"))==NULL)
    {printf("output file error\n");
    exit(0);}

for (x=5; x<xmax-6; x=x+5)
    {for (y=5; y<ymax-6; y=y+5)
    {
    xcomponent=velfieldx[x-2][y+2]+velfieldx[x-
1][y+2]+velfieldx[x][y+2]+velfieldx[x+1][y+2]+velfieldx[x+2][y+2]
+velfieldx[x-2][y+1]+velfieldx[x-
1][y+1]+velfieldx[x][y+1]+velfieldx[x+1][y+1]+velfieldx[x+2][y+1]
+velfieldx[x-2][y] +velfieldx[x-1][y] +velfieldx[x][y]
+velfieldx[x+1][y] +velfieldx[x+2][y]
+velfieldx[x-2][y-1]+velfieldx[x-1][y-1]+velfieldx[x][y-
1]+velfieldx[x+1][y-1]+velfieldx[x+2][y-1]
+velfieldx[x-2][y-2]+velfieldx[x-1][y-2]+velfieldx[x][y-
2]+velfieldx[x+1][y-2]+velfieldx[x+2][y-2];

    ycomponent=velfielgy[x-2][y+2]+velfielgy[x-
1][y+2]+velfielgy[x][y+2]+velfielgy[x+1][y+2]+velfielgy[x+2][y+2]
+velfielgy[x-2][y+1]+velfielgy[x-
1][y+1]+velfielgy[x][y+1]+velfielgy[x+1][y+1]+velfielgy[x+2][y+1]
+velfielgy[x-2][y] +velfielgy[x-1][y] +velfielgy[x][y]
+velfielgy[x+1][y] +velfielgy[x+2][y]
}
}
}

```

```
        +velfieldy[x-2][y-1]+velfieldy[x-1][y-1]+velfieldy[x][y-
1]+velfieldy[x+1][y-1]+velfieldy[x+2][y-1]
        +velfieldy[x-2][y-2]+velfieldy[x-1][y-2]+velfieldy[x][y-
2]+velfieldy[x+1][y-2]+velfieldy[x+2][y-2];

        fprintf(output1,"%5i %5i %3.7f %3.7f\n", x, y, xcomponent,
ycomponent);}
    }

/*-----*/

//Final operations
fclose(output);
fclose(output1);
getch();
return 0;
} //The end of the main part of the algorithm

/*-----*/
```

```

/*FHP-1 LGCA for fluid flow in porous medium simulation.Graphical outputs*/

//Definition of standard library functions
#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/*-----*/

/* Beginning of a main part of the program */

int main(void)
{

//Declaration of variables
float xcomponent, ycomponent, step=17.5;
int x, y, xstart, ystart, xend, yend;
FILE *output1;
/*-----*/
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    int xmax, ymax;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "C:\\TC\\BGI");

    /* read result of initialization */
    errorcode = graphresult();
    /* an error occurred */
    if (errorcode != grOk)
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
/*-----*/
    setcolor(getmaxcolor());
    setbkcolor(15);
    xmax = getmaxx();
    ymax = getmaxy();
/*-----*/

//Opening the data file VELFIEL.CPP. Data graphical representation.

//Open file, testing for seccess
if((vystup1=fopen("C:\\Outputs\\Filter\\Graphic\\Velfiel1.cpp", "r"))==NULL)
    {printf("input vfldx file error\n");exit(0);}

while(fscanf(output1,"%i %i %f %f\n", &x,&y,&xcomponent,&ycomponent)!=EOF)
    {xstart=1.4*x; ystart=1.4*y;
    xend=step*xcomponent; yend=step*ycomponent;

    if (pow(xstart,2)+pow(ystart,2)>0)
        {if(x/1==x/1.)
        {if(y/1==y/1.)
            {setcolor(12); if((xkon<0)|| (ykon<0)){setcolor(2);}
            line(xstart, ystart, xend+xend, yend+yend);
            putpixel(xstart, ystart,1);}
        }}}
}

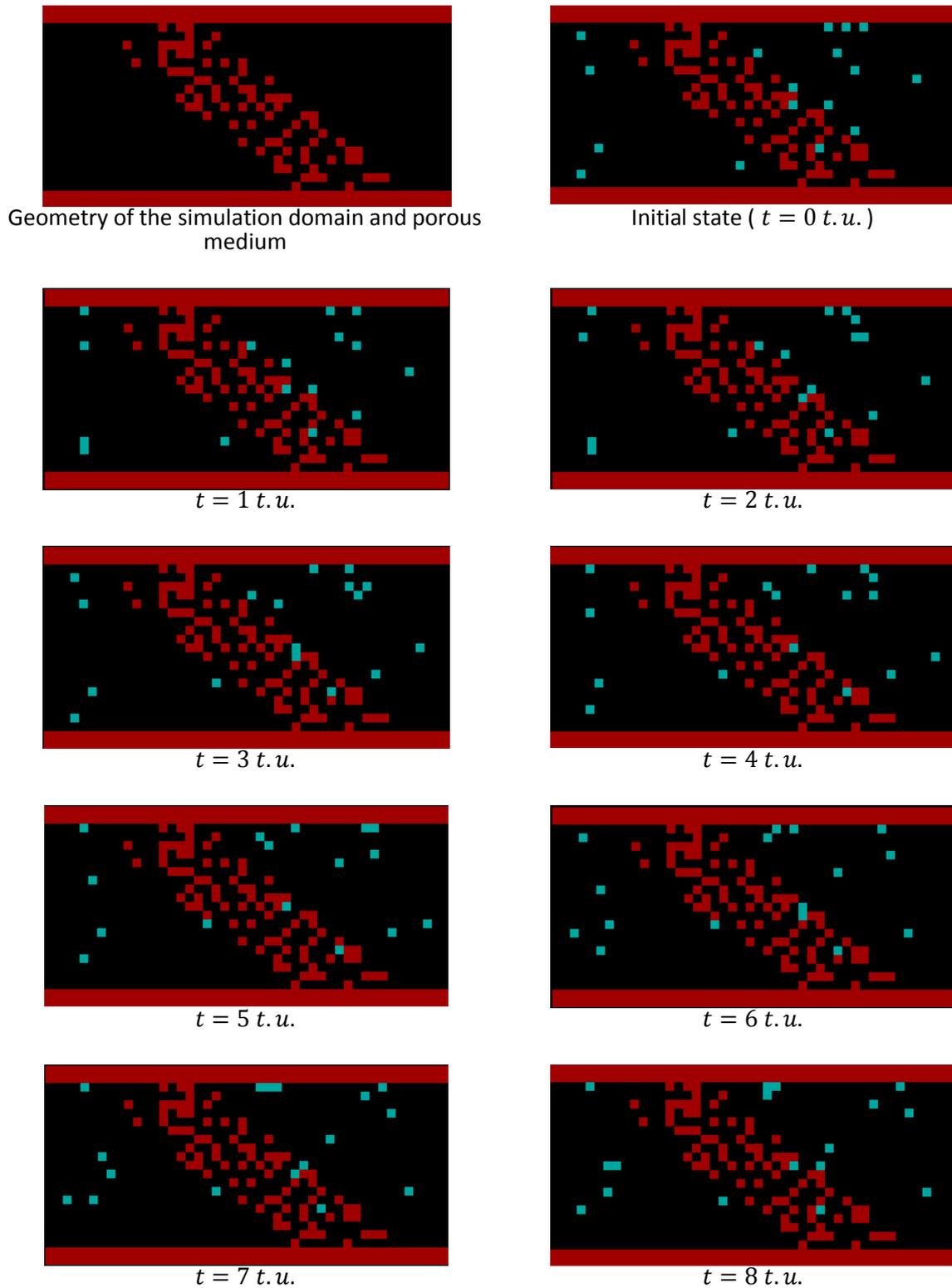
```

```
    }  
  
/*-----*/  
  
//Final operations  
fclose(output1);  
getch();  
closegraph();  
return 0;  
} //The end of the main part of the algorithm  
  
/*-----*/
```

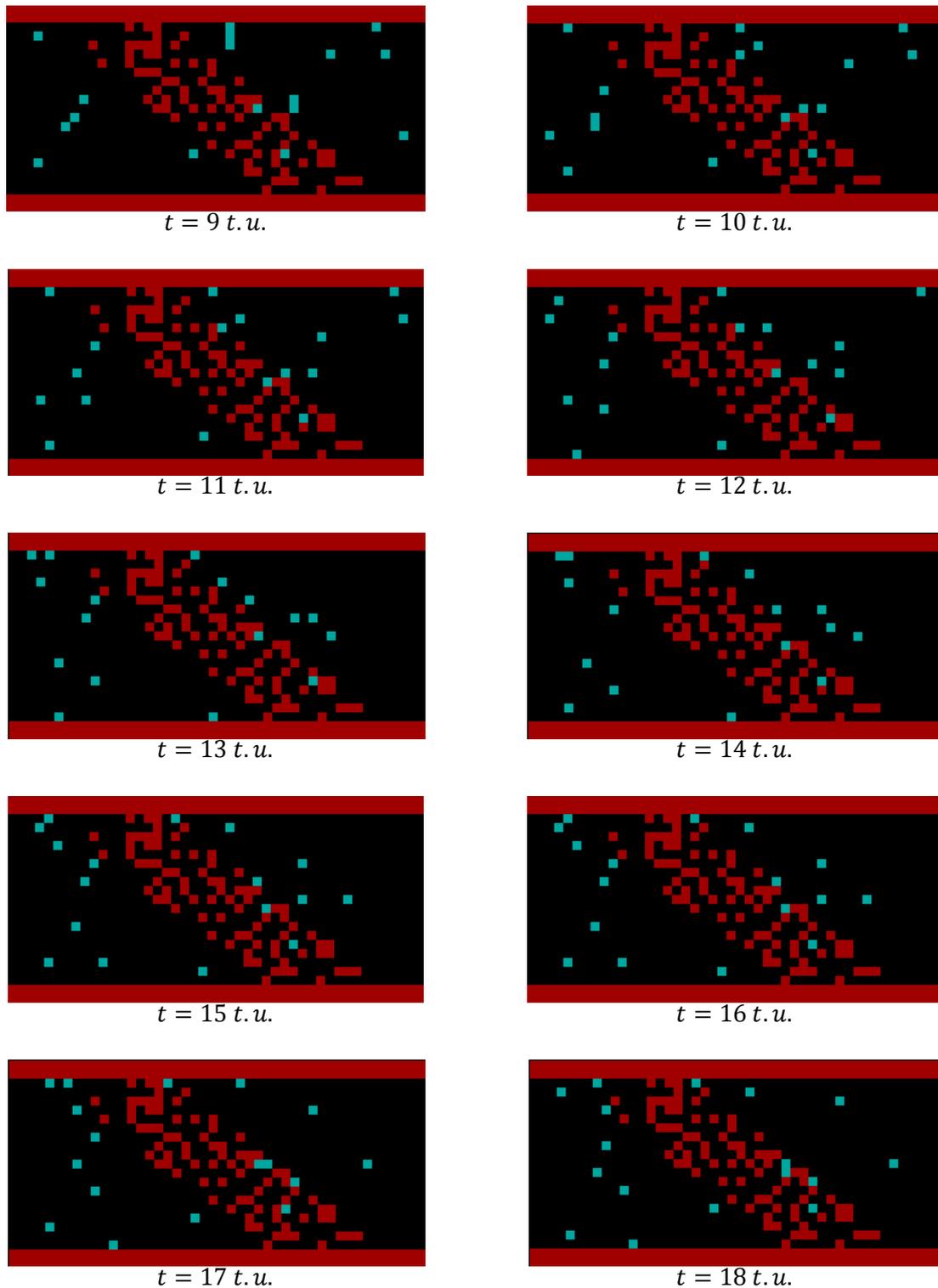
**APPENDIX P**

Computer simulation of the fluid flow through declined porous media. Evolution in time for a period of 20 time steps.

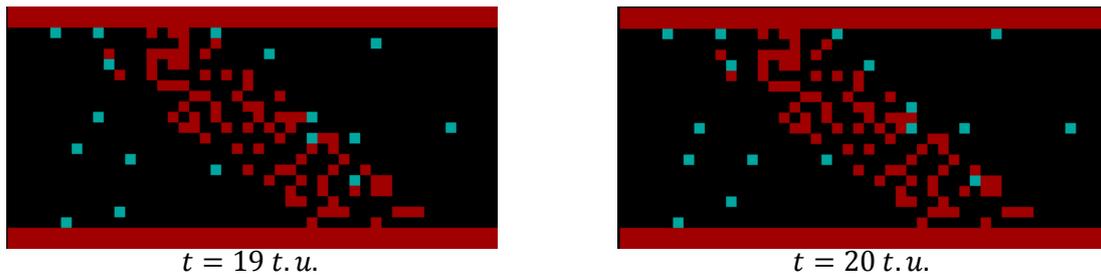
---



**Figure P-1:** Computer simulation of the fluid flow through the declined porous medium presented on the reduced simulation domain of a size  $N_x \times N_y = 48 l.u. \times 25 l.u.$  The lattice gas average density is  $0,2 m.u./l.u.$ , porosity of the random generated porous structure is  $0,7$ . Movement of fluid particles is monitored for a period of 20 time steps with an interval of  $1 t.u.$



**Figure P-1 (continuation):** Computer simulation of the fluid flow through the declined porous medium presented on the reduced simulation domain of a size  $N_x \times N_y = 48 l.u. \times 25 l.u.$  The lattice gas average density is  $0,2 m.u./l.u.$ , porosity of the random generated porous structure is  $0,7$ . Movement of fluid particles is monitored for a period of 20 time steps with an interval of  $1 t.u.$

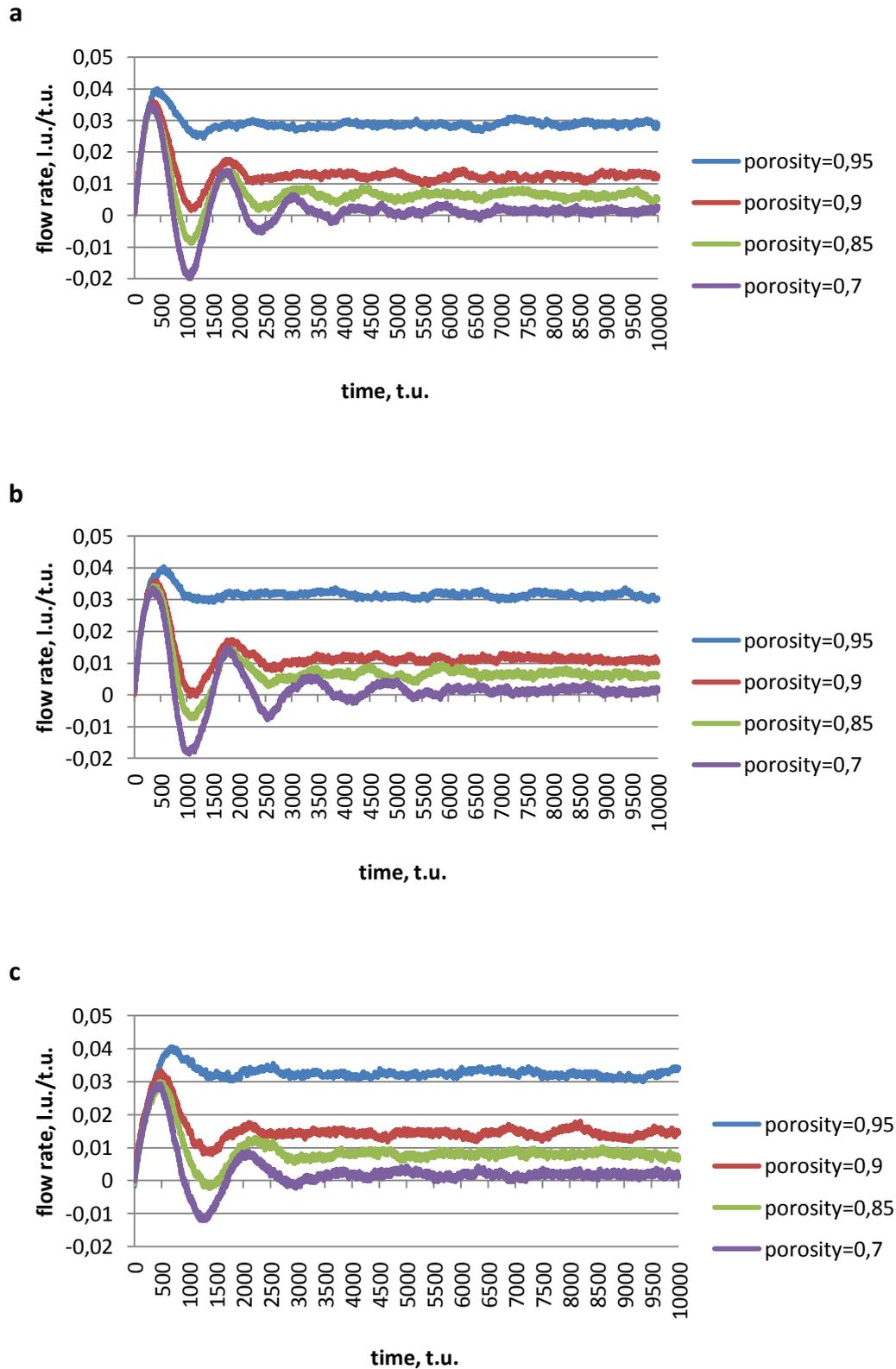


**Figure P-1 (continuation):** Computer simulation of the fluid flow through the porous medium presented on the reduced simulation domain of a size  $N_x \times N_y = 48 \text{ l.u.} \times 25 \text{ l.u.}$ . The lattice gas average density is  $0,2 \text{ m.u./l.u.}$ , porosity of the random generated porous structure is  $0,7$ . Movement of fluid particles is monitored for a period of 20 time steps with an interval of  $1 \text{ t.u.}$

**APPENDIX Q**

Computer simulation of the fluid flow through declined porous media. Flow rate as a function of time for various inclination angle  $\alpha$ .

---

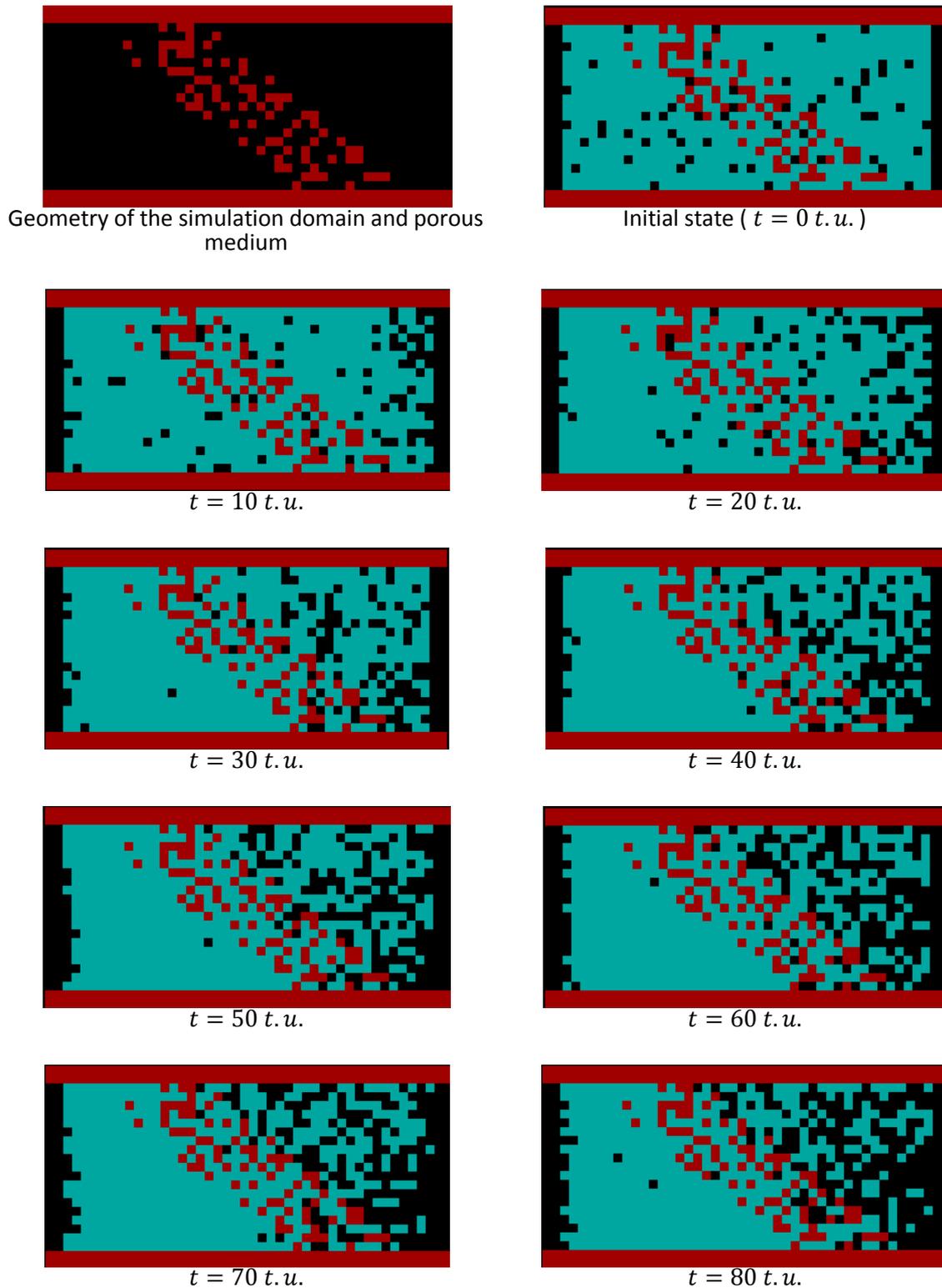


**Figure Q-1:** The flow rate as a function of time for various porosity and inclination angle  $\alpha = 15^\circ$  (a),  $\alpha = 35^\circ$  (b) and  $\alpha = 55^\circ$  (c). The length  $L$  of the channel 450 l. u., the width  $d$  is  $250 \frac{\sqrt{3}}{2}$  l. u.

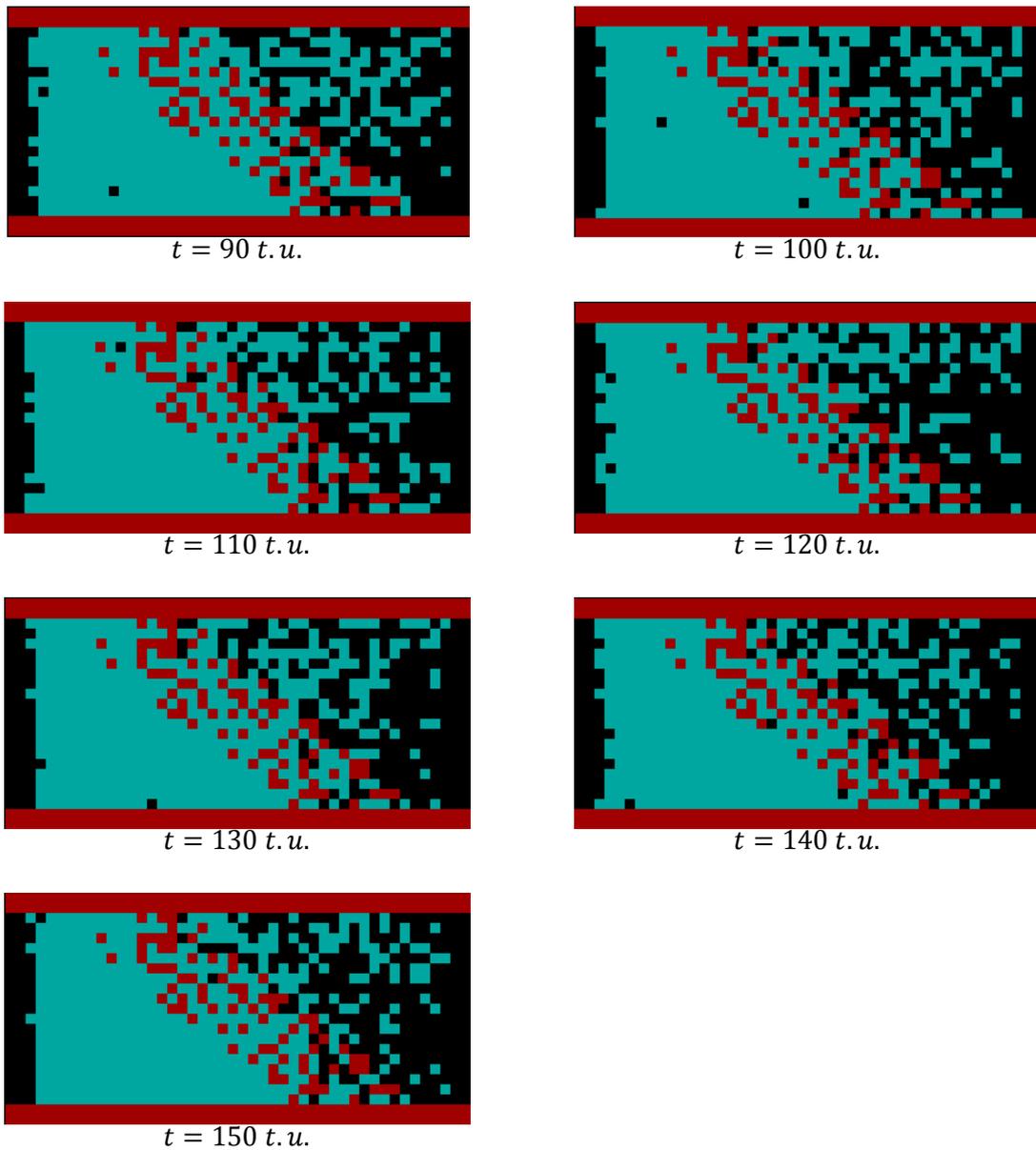
**APPENDIX R**

Computer simulation of the fluid flow through declined porous media. Time evolution of the system with reduced simulation domain.

---



**Figure R-1:** Computer simulation of the fluid flow through the porous medium presented on the reduced simulation domain of a size  $N_x \times N_y = 48 l.u. \times 25 l.u.$  The lattice gas average density is  $3,5 m.u./l.u.$ , porosity of the random generated porous structure is  $0,7$ . Fluid flow is monitored for a period of 150 time steps with an interval of  $10 t.u.$

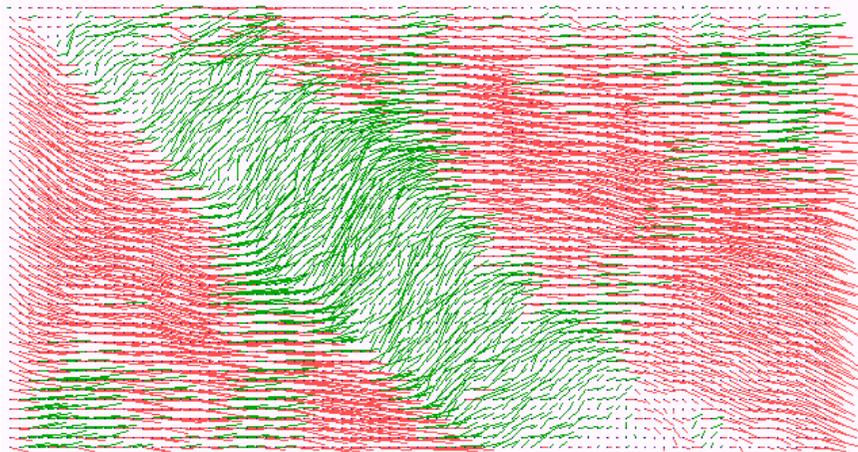


**Figure R-1 (continuation):** Computer simulation of the fluid flow through the porous medium presented on the reduced simulation domain of a size  $N_x \times N_y = 48 \text{ l.u.} \times 25 \text{ l.u.}$ . The lattice gas average density is  $3,5 \text{ m.u./l.u.}$ , porosity of the random generated porous structure is  $0,7$ . Fluid flow is monitored for a period of 150 time steps with an interval of  $10 \text{ t.u.}$

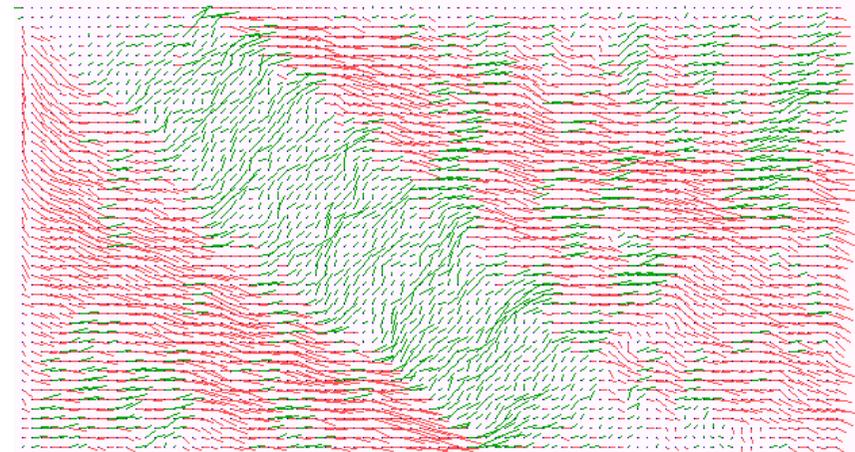
## APPENDIX S

Computer simulation of the fluid flow through  
declined porous media. Fields of velocity vectors.

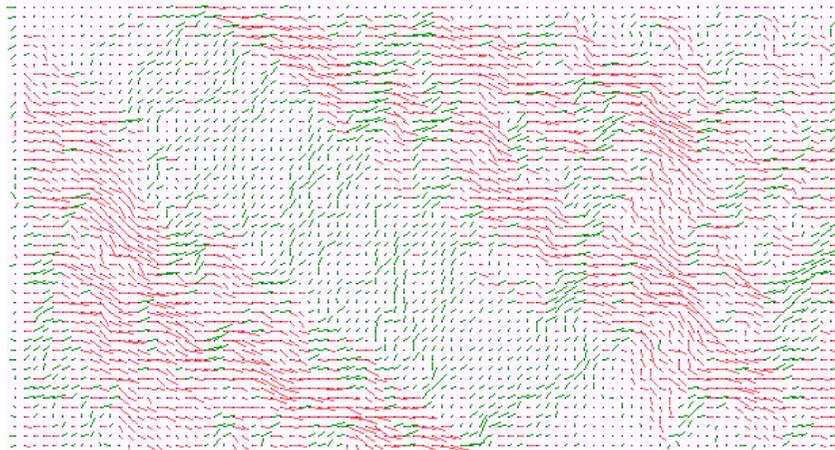
---



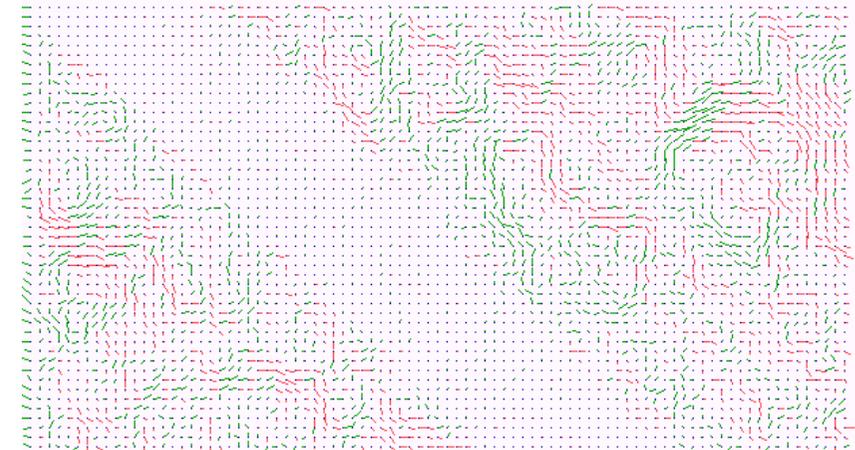
Porosity is 0,95



Porosity is 0,9



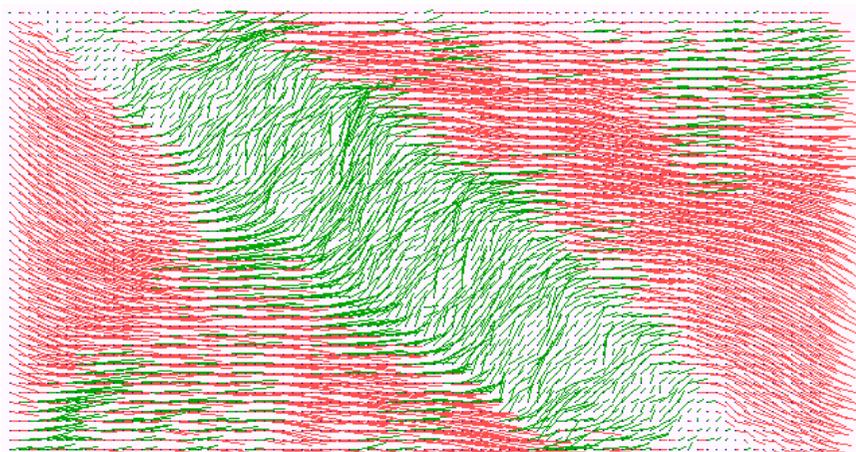
Porosity is 0,85



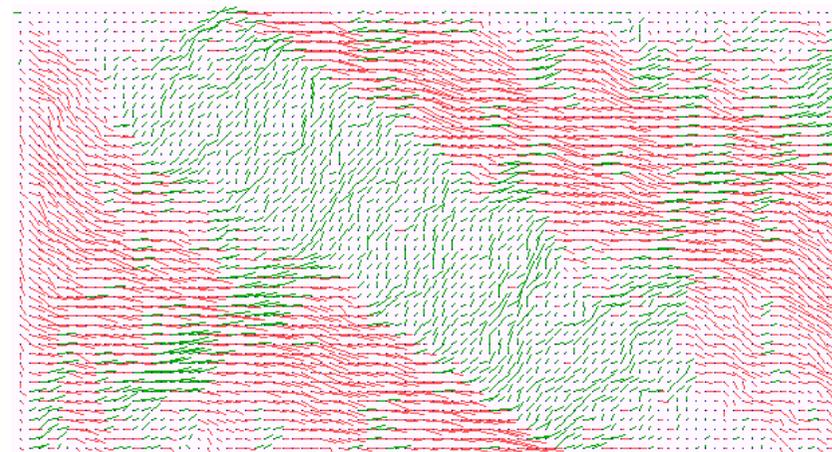
Porosity is 0,7

**Figure S-1:** Fluid velocity directions inside the channel and declined porous media of various porosity. The inclination angle  $\alpha$  is  $15^\circ$

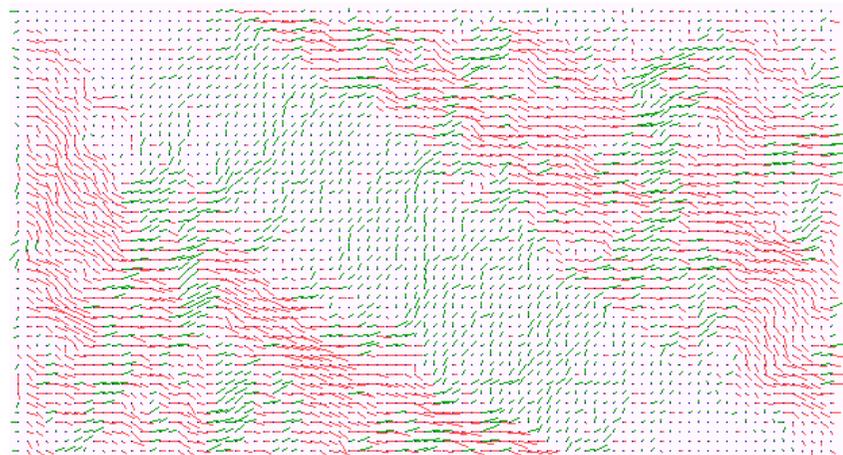
Inclination angle  $\alpha=35^\circ$



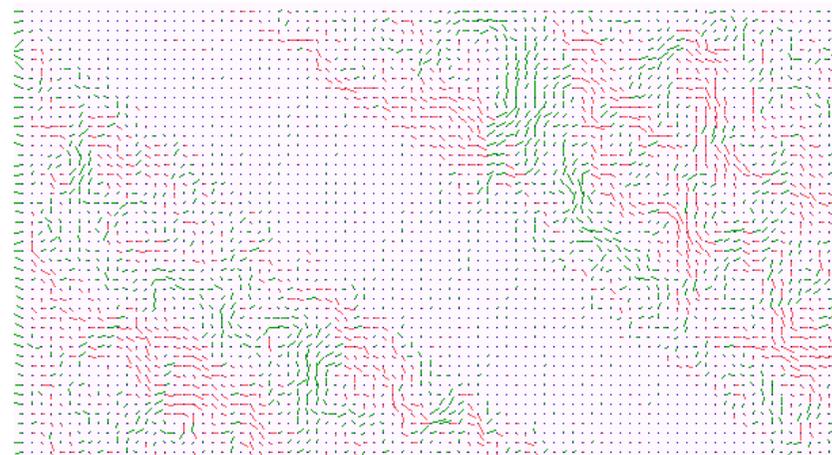
Porosity is 0,95



Porosity is 0,9

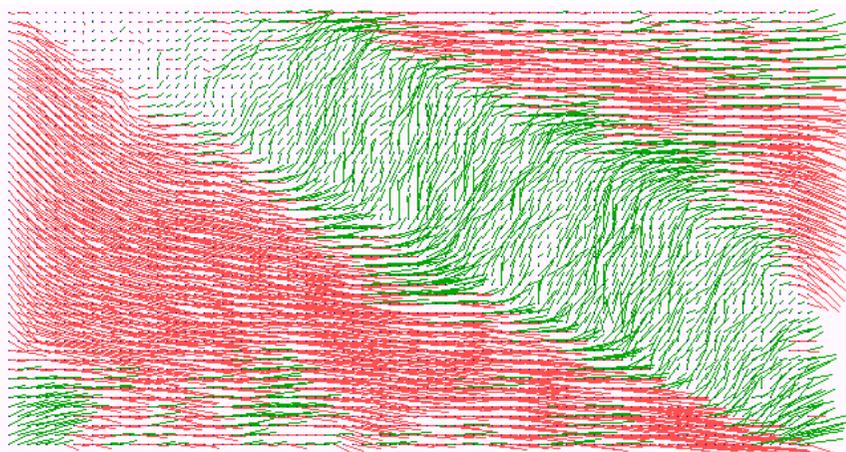


Porosity is 0,85

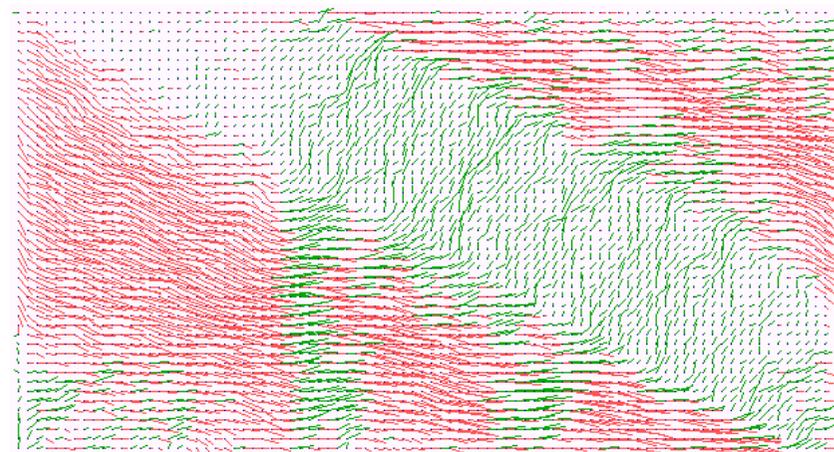


Porosity is 0,7

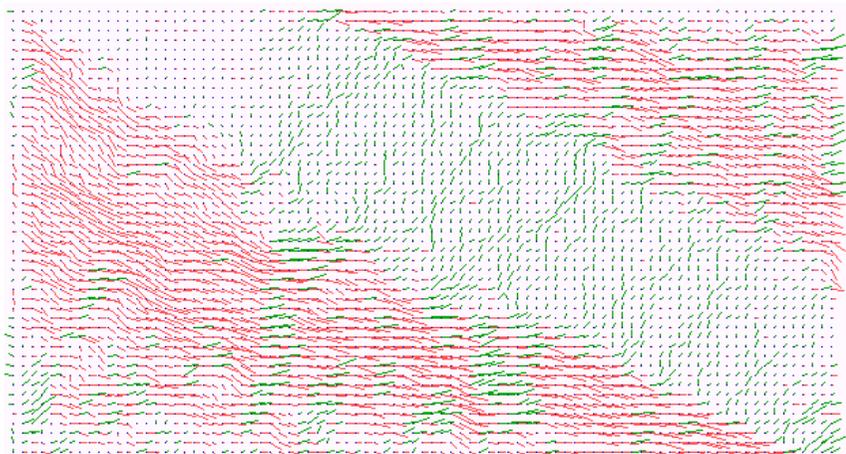
**Figure S-2:** Fluid velocity directions inside the channel and declined porous media of various porosity. The inclination angle  $\alpha$  is  $35^\circ$



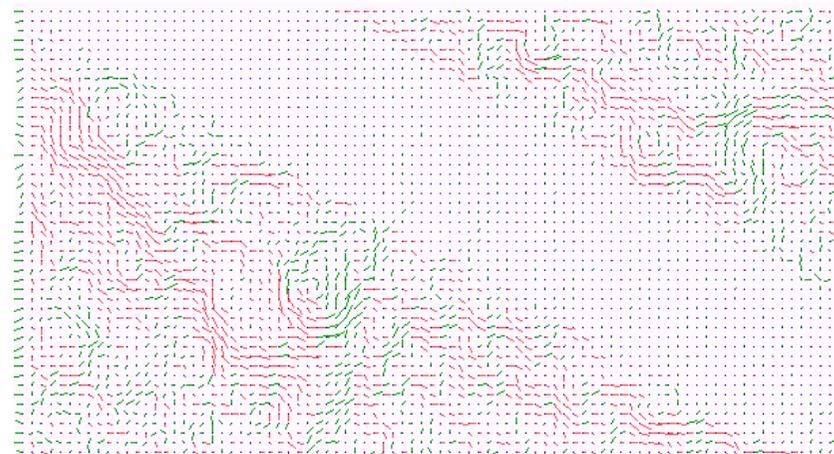
Porosity is 0,95



Porosity is 0,9



Porosity is 0,85



Porosity is 0,7

**Figure S-3:** Fluid velocity directions inside the channel and declined porous media of various porosity. The inclination angle  $\alpha$  is  $55^\circ$