Visualization and Mesoscopic Simulation in Systems Biology

Von der Fakultät Informatik, Elektrotechnik und
Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Martin Samuel Falk

aus Hechingen

Hauptberichter:	Prof. Dr. Thomas Ertl
Mitberichter:	Prof. Raghu Machiraju, Ph.D.
	Prof. Dr. Peter Scheurich
Tag der mündlichen Prüfung:	29. April 2013

Visualisierungsinstitut
der Universität Stuttgart

2013

In biology, nothing is clear, everything is too complicated, everything is a mess, and just when you think you understand something, you peel off a layer and find deeper complications beneath. Nature is anything but simple.

Richard Preston, The Hot Zone

Acknowledgments

First of all, I especially want to thank my advisor Thomas Ertl, who guided and supported my work and gave me the opportunity to pursue my doctoral studies at VIS and VISUS. I am also grateful to Raghu Machiraju and Peter Scheurich for agreeing to act as referees for this thesis.

It has been a great pleasure to cooperate with Michael Klann and I want to thank him for our excellent collaboration in the past years. In addition, I had the pleasure to collaborate with Markus Daub, Sebastian Grottel, Michael Krone, Filip Sadlo, and Markus Üffinger over the recent years. I also enjoyed the time with my office mates Christiane Taras, Alexandros Panagiotidis, Daniel Kauker, Gustavo Mallo Machado, and Benjamin Höferlin. For the many fruitful and sometimes absurd but nonetheless amusing discussions I am thankful to Benjamin Höferlin, Markus Höferlin, Steffen Müthing, Harald Sanftmann, Markus Üffinger, and Michael Wörner.

I would also like to thank the remaining members of VIS and VISUS for the great time I had in Stuttgart. Many thanks go to Michael Klann, Julia Möhrmann, Steffen Koch, and Michael Wörner for proof-reading my thesis. Moreover, I thank the students I supervised and with whom I worked with over the last years—in alphabetical order: Hendrik Hochstetter, Michael Ott, Ana Cristina Pintilie, and Mikael Vaaraniemi.

Financial support was provided by the Center Systems Biology (CSB) at the University of Stuttgart and the state of Baden Württemberg. Thanks go to Heinz Köppl for organizing the stay at the Eidgenössische Technische Hochschule (ETH) in Zürich, Switzerland, and to the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart for financing it. My work benefited greatly from the time at the ETH, which also provided insights into different research facilities.

Above all, I want to express my deepest gratitude to Judith and my family for supporting and motivating me during this work.

Stuttgart,
May 2013

Martin Falk

Contents

List of Abbreviations and Acronyms

ADP	adenosine diphosphate ($C_{10}H_{15}N_5O_{10}P_2$)
ATP	adenosine triphosphate ($C_{10}H_{16}N_5O_{13}P_3$)
BD	Brownian dynamics
CLSM	confocal laser scanning microscopy
CoC	circle of confusion
CPU	central processing unit
CSB	Center Systems Biology at the University of Stuttgart
CUDA	Compute Unified Device Architecture developed by NVIDIA
DNA	deoxyribonucleic acid
DoF	depth of field
DVR	direct volume rendering
EM	electron microscopy
ER	endoplasmatic reticulum
ERK	extracellular-signal-regulated kinase
GFP	green fluorescent protein
GLSL	OpenGL shading language
GPGPU	general purpose computing on graphics processing units
GPU	graphics processing unit
MAP	mitogen-activated protein
MAPK	mitogen-activated protein kinase
MAP2K	MAPK kinase (also MAPKK)
MAP3K	MAP2K kinase (also MAPKKK)
MC	Monte Carlo
MD	molecular dynamics
MIMD	multiple instruction, multiple data
MISD	multiple instruction, single data
MTOC	microtubule-organizing center
NURBS	non-uniform rational B-spline
ODE	ordinary differential equation
OpenCL	open compute language
OpenGL	open graphics libray
OpenMP	open multi-processing[1]
OSAO	object-space ambient occlusion

[1] http://www.openmp.org

RCSB PDB	protein data base of the Research Collaboratory for Structural Bioinformatics (RCSB)[2]
PDE	partial differential equation
pixel	picture element
RAM	random access memory
SBML	systems biology mark-up language
SBGN	systems biology graphical notation[3]
SDE	stochastic differential equation
SES	solvent excluded surface
SIMD	single instruction, multiple data
SIMT	single instruction, multiple threads
SIMS	single instruction, multiple streams
SISD	single instruction, single data
SPH	smoothed particle hydrodynamics
SSA	stochastic simulation algorithm
SSAO	screen-space ambient occlusion
texel	texture picture element (cf. **pixel**)
TNF	tumor necrosis factor
voxel	volume picture element (cf. **pixel**)

Units

billion	thousand million (10^9)
Da	dalton, defined as one twelfth the mass of carbon-12 (^{12}C), $1.660\,538\,921 \times 10^{-27}$ kg
fps	frames per second
GiB	gibibyte, 2^{30} bytes
KiB	kibibyte, 2^{10} bytes
MiB	mebibyte, 2^{20} byte

[2] http://www.pdb.org
[3] http://www.sbgn.org

List of Symbols

Symbol	Unit	Explanation
γ_{ij}	$\mu m^3/s$	diffusion limit
κ_{ij}	L/(mol s)	microscopic reaction rate constant for second-order reactions
ρ	kg/m^3	density
τ	s	reaction delay
ξ		random number (uniformly distributed), $\xi \sim \mathcal{U}(0, 1)$
$\boldsymbol{\xi}_{\mathcal{U}}$		random vector (uniformly distributed), $\boldsymbol{\xi}_{\mathcal{U}} \sim \boldsymbol{\mathcal{U}}(-\sqrt{3} \cdot \mathbf{1}, \sqrt{3} \cdot \mathbf{1})$
$\boldsymbol{\xi}_{\mathcal{N}}$		random vector (normally distributed), $\boldsymbol{\xi}_{\mathcal{N}} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, I)$
a		lens aperture, also f-number, defined as ratio of the focal length f and the aperture diameter d_a, i.e. $a = f/d_a$
c_X	mol/L	molar concentration of molecule X
D	m^2/s	diffusion coefficient
f	m	focal length of a lens
k_{cat}	1/s	unimolecular rate constant (turn-over number)
k_i	1/s	reaction rate constant for first-order reactions
k_{ij}	L/(mol s)	macroscopic reaction rate constant for second-order reactions
K_m	mol/L	Michaelis constant in enzyme kinetics
N_A	1/mol	Avogadro constant: $N_A = 6.022\,141\,29 \times 10^{23}\,mol^{-1}$
$\boldsymbol{p}$		position or vector, $\boldsymbol{p} \in \mathbb{R}^3$
P_i		reaction probability of a first-order reaction
P_{ij}		reaction probability of a second-order reaction
r	µm	radius
r_{mem}	µm	radius of plasma membrane
r_{nucl}	µm	radius of nucleus
s_{fil}	µm	extent of binding region around cytoskeletal filaments
s_{mem}	µm	extent of binding region at membranes
t	s	time
Δt	s	time interval
V	μm^3	volume
v	µm/s	velocity

Abstract

Biological cells appear everywhere on earth. They might live on their own as unicellular organism, like bacteria, or might form complex organisms consisting of several thousands or millions of cells. Despite their small size of only a few micrometers, they are complex little miracles. A better understanding of the internal mechanisms and interplays within a single cell is the key to the understanding of life. In the context of this thesis, the mechanism of cellular signal transduction, i.e. relaying a signal from outside the cell by different means of transport toward its target inside the cell, is employed. Understanding and adjusting parts of the cellular signaling mechanism will eventually lead to the design of new drugs with less or even without any side-effects. Besides experiments, understanding can also be achieved by numerical simulations of cellular behavior. This is where systems biology closely relates and depends on recent research results in computer science in order to deal with the modeling, the simulation, and the analysis of the computational results.

Since a single cell can consist of billions of molecules, the simulation of intracellular processes requires a simplifiedmodel. Typically, mesoscopic models are used containing only parts that are believed to be necessary for the processes. The simulation domain has to be three dimensional to consider the spatial, possibly asymmetric, intracellular architecture filled with individual particles representing signaling molecules. In contrast to continuous models defined by systems of partial differential equations, a particle-based model allows tracking individual molecules moving through the cell. This particle-based approach, however, demands for a higher computational effort than, e.g., non-spatial models that can be solved with ordinary differential equations. The overall process of signal propagation usually requires between minutes and hours to complete, but the movement of molecules and the interactions between them have to be computed in the range of microseconds. Hence, the computation of thousands of consecutive time steps is necessary, requiring several hours or even days of computational time for a sequential simulation. The need for several simulation runs with different parameter settings, a higher level of detail including more particles, or a generally more precise simulation, i.e. smaller time steps, demands for short execution times of the simulation. To speed up the simulation, the parallel hardware of current central processing units (CPUs) and graphics processing units (GPUs) can be employed. The parallelization of interacting particles, however, is non-trivial and requires special care when utilizing modern many-core architectures like the GPU. Finally, the resulting data has to be analyzed by domain experts and, therefore, has to be represented in meaningful ways. Typical prevalent analysis methods include the aggregation of the data in tables or simple 2D graph plots, sometimes 3D plots for continuous data. Despite the fact that techniques for the interactive visualization of data in 3D are well-known, so far none of the methods have been applied to the biological context of single cell models and specialized visualizations fitted to the experts' need are missing. Another issue is the hardware available to the domain experts that can be used for the task of visualizing the increasing amount of time-dependent data

resulting from simulations. Exa-scale visualization still seems to be a long way off, but even the data sets available today are pushing the rendering capability of current graphics hardware to its limits. However, it is important that the visualization keeps up with the simulations to ensure that domain experts can still analyze their data sets. To deal with the massive amount of data to come, compute clusters will be necessary with specialized hardware dedicated to data visualization. It is, thus, important, to develop visualization algorithms for this dedicated hardware, which is currently available as GPU.

In this thesis, the computational power of recent many-core architectures (CPUs as well as GPUs) is harnessed for both the simulation and the visualizations. Novel parallel algorithms are introduced to parallelize the spatio-temporal, mesoscopic particle simulation to fit the architectures of CPU and GPU in a similar way, easing the portability between both. Besides molecular diffusion, the simulation considers extracellular effects on the signal propagation as well as the import of molecules into the nucleus and a dynamic cytoskeleton. An extensive comparison between different configurations is performed leading to the conclusion that the usage of GPUs is not always beneficial. Depending on the simulation setup, however, the GPU implementation can be up to ten times faster than the parallel simulation on the CPU. For the visual data analysis, novel interactive visualization techniques were developed to visualize the 3D simulation results. Existing glyph-based approaches are combined in a new way facilitating the visualization of the individual molecules in the interior of the cell as well as their trajectories. A novel implementation of the depth of field effect, as known from photography, combined with additional depth cues and coloring aid the visual perception and reduce visual clutter. To obtain a continuous signal distribution from the discrete particles, techniques known from volume rendering are employed. The visualization of the underlying atomic structures provides new detailed insights and can be used for educational purposes besides showing the original data. The proposed technique allows for the interactive visualization of data sets containing several billion atoms per time step. A microscope-like visualization allows for the first time to generate images of synthetic data similar to images obtained in wet lab experiments. The simulation and the visualizations are merged into a prototypical framework, thereby supporting the domain expert during the different stages of model development, i.e. design, parallel simulation, and analysis.

Although the proposed methods for both simulation and visualization were developed with the study of single-cell signal transduction processes in mind, they are also applicable to models consisting of several cells and other particle-based scenarios. Examples in this thesis include the diffusion of drugs into a tumor, the detection of protein cavities, and molecular dynamics data from laser ablation simulations, among others.

German Abstract
—Zusammenfassung—

Zellen kommen auf der Erde überall vor. Sie leben entweder auf sich selbst gestellt als Einzeller, wie beispielsweise Bakterien, oder bilden zusammen mit tausenden von Zellen hochkomplexe Organismen. Trotz ihrer Größe von nur wenigen Mikrometern sind sie dennoch sehr komplexe Wunderwerke. Ein besseres Verständnis der komplizierten, internen Mechanismen und der Wechselwirkungen innerhalb einer einzelnen Zelle kann als Voraussetzung gesehen werden, um das Leben an sich verstehen zu können. Als ein Beitrag dazu beschäftigt sich diese Dissertation mit dem Mechanismus der zellulären Signaltransduktion. Hierbei wird ein Signal, das seinen Ursprung außerhalb der Zelle hat, über verschiedene Transportwege zu seinem Ziel, dem Zellkern, übermittelt. Das Verstehen und Anpassen von Teilen dieses Signalübertragungsmechanismus kann letztendlich zur Entwicklung von neuen Medikamenten beitragen, die nur geringe oder keine Nebenwirkungen aufweisen. Neben Experimenten im Labor können auch numerische Simulationen von zellulärem Verhalten genutzt werden, um neue Einsichten und ein besseres Verständnis zu gewinnen. Solche Simulationen setzen jedoch voraus, dass theoretische Modelle entworfen und ausgewertet werden. An diesem Punkt wird klar, dass die Systembiologie von aktuellen Forschungsergebnissen der Informatik profitiert und auch abhängig ist, um die Modellierung, die Simulation und die Analyse der berechneten Ergebnisse zu bewältigen.

Die Simulation intrazellulärer Prozesse erfordert heutzutage noch ein vereinfachtes, mesoskopisches Zellmodell, da eine einzelne Zelle aus mehreren Milliarden Atomen bestehen kann. Um die räumliche, möglicherweise auch asymmetrische Architektur der Zelle zu berücksichtigen, müssen sowohl das Simulationsgebiet als auch die Simulation dreidimensional sein. Eine partikelbasierte Simulation ermöglicht im Gegensatz zu kontinuierlichen Modellen – gegeben durch ein System von partiellen Differentialgleichungen – das Verfolgen einzelner Partikel. Diese Partikel stellen Signalmoleküle auf ihrem Weg durch die Zelle dar. Solche partikelbasierten Simulationen erfordern allerdings deutlich mehr Berechnungsaufwand als beispielsweise nicht-räumliche Modelle, die mittels gewöhnlicher Differentialgleichungen gelöst werden können. Der gesamte Prozess einer Signalübertragung innerhalb der Zelle benötigt in der Realität zwischen wenigen Minuten und mehreren Stunden. Die Bewegungen der Moleküle sowie die Interaktionen zwischen ihnen müssen jedoch mit einer Zeitauflösung von wenigen Mikrosekunden berücksichtigt werden. Daraus folgt, dass die Auswertung von tausenden aufeinanderfolgenden Zeitschritten notwendig ist, deren Berechnung mit einer sequentiellen Simulation einige Stunden oder sogar Tage benötigen würde. Wünschenswert sind kurze Simulationszeiten, die oft auch notwendig sind – insbesondere wenn verschiedene Parametersätze analysiert, mehr Details simuliert oder einfach die Simulationen zeitlich feiner aufgelöst werden sollen. Um die Simulationen zu beschleunigen, kann die parallele Hardware heutiger Hauptprozessoren (CPUs) und Grafikprozessoren (GPUs) eingesetzt werden. Die Parallelisierung von interagierenden Partikeln ist jedoch nicht trivial und erfordert daher

spezielle Mittel und Wege, wenn moderne Mehrkernprozessoren – wie beispielsweise GPUs – verwendet werden. Im Anschluss an die Simulation müssen die resultierenden Daten von Experten analysiert werden und sollten daher möglichst aussagekräftig dargestellt werden. Zu den gängigen Analyseverfahren zählt neben der Ansammlung der Daten in Tabellen auch die Darstellung in einfachen 2D-Liniendiagrammen und für kontinuierliche Daten darüber hinaus als dreidimensionale Graphen. Obwohl Techniken für die interaktive Visualisierung von 3D-Daten bekannt sind, wurde bislang keine der Methoden im Kontext eines biologischen Zellmodells eingesetzt. Speziell an die Ansprüche der Experten angepasste Visualisierungen fehlen. Ein weiteres Problem stellt die den Experten zur Verfügung stehende Hardware dar, welche für die Visualisierung der stets zunehmenden Menge an zeitabhängigen Simulationsdaten verwendet werden kann. Die Visualisierungen im Exa-Bereich scheinen noch weit in der Zukunft zu liegen, aber selbst für heutzutage verfügbare Datensätze stoßen aktuelle Grafikkarten immer öfter an die Grenzen der maximalen Darstellungsleistung. Es ist jedoch von großer Bedeutung, dass die Visualisierung mit der Entwicklung der Simulationen Schritt hält, um sicherzustellen, dass Experten ihre Daten weiterhin analysieren können. Um die zukünftigen enormen Datenmengen zu bewältigen, wird der Einsatz von Compute Clusters notwendig sein, die über eine spezialisierte Hardware für die Datenvisualisierung verfügen. Daher ist die Entwicklung von Visualisierungsalgorithmen für so eine spezialisierte Hardware wichtig, die heute in Form von GPUs bereits verfügbar ist.

In dieser Dissertation wird die Rechenleistung der jüngsten Architekturen von Mehrkernprozessoren (sowohl CPUs als auch GPUs) für die Simulation wie auch für die Visualisierung eingesetzt. Neue parallele Algorithmen werden vorgestellt, die zur Parallelisierung der räumlich-zeitlichen, mesoskopischen Partikelsimulation auf CPU und GPU in ähnlicher Weise eingesetzt werden können, um die Portabilität zwischen beiden Architekturen zu vereinfachen. Neben der molekularen Diffusion werden in der Simulation auch folgende Mechanismen berücksichtigt: extrazelluläre Effekte auf die Signalübertragung, Import von Molekülen in den Zellkern und dynamisches Zellskelett. Ein ausführlicher Vergleich, durchgeführt für verschiedene Simulationskonfigurationen, führt zu dem Schluss, dass die Verwendung der GPU nicht in allen Fällen nutzbringend ist. In Abhängigkeit von den Simulationsparametern kann die GPU-Implementierung in Einzelfällen jedoch bis zu zehn Mal schneller sein als die parallele Simulation auf der CPU. Für die visuelle Datenanalyse der 3D-Simulationsergebnisse wurden neue interaktive Visualisierungstechniken entwickelt. Bereits existierende, glyphbasierte Ansätze wurden in einer neuen Weise kombiniert, um die Visualisierung der individuellen Moleküle und ihre Trajektorien im Inneren der Zelle zu ermöglichen. Der Effekt der Schärfentiefe, wie er auch in der Fotografie bekannt ist, kann mit zusätzlichen Tiefenhinweisen und Einfärbungen kombiniert werden, um die visuelle Wahrnehmung zu unterstützen und die Überreizung durch zuviele dargestellte Elemente zu reduzieren. Techniken für die Darstellung von Volumendaten werden in leicht abgewandelter Form zur Visualisierung einer kontinuierlichen Signalverteilung eingesetzt, die aus den diskreten Partikeln berechnet wird. Die Visualisierung der Atomstrukturen der einzelnen Moleküle bietet neue Einsichten in das zelluläre Innenleben und kann neben der Darstellung der ursprünglichen Daten auch für pädagogische Zwecke

eingesetzt werden. Der in dieser Arbeit vorgestellte Ansatz ermöglicht die interaktive Visualisierung von Datensätzen, die aus mehreren Milliarden Atomen pro Zeitschritt bestehen. Eine mikroskop-ähnliche Visualisierung erlaubt zum ersten Mal die Generierung von Bildern aus synthetischen Daten, die denen aus Laborversuchen nur wenig nachstehen. Die Simulation und die verschiedenen Visualisierungen wurden in einem prototypischen System vereinigt. Dieses System ist in der Lage, den Experten im Verlauf der verschiedenen Stufen der Modellentwicklung – d.h. Entwurf, parallele Simulation und Analyse – zu unterstützen.

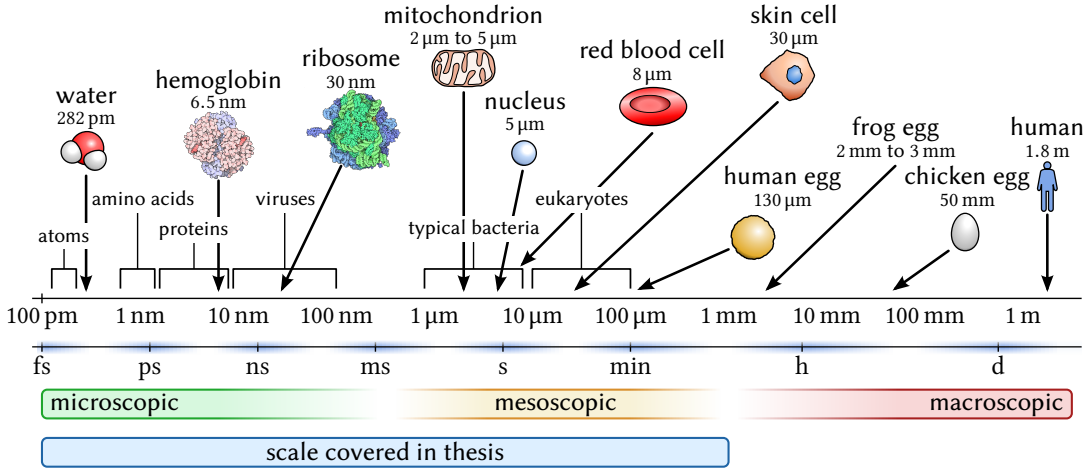
Obwohl die vorgestellten Simulations- und Visualisierungsmethoden mit dem Ziel, die Signalübertragungsprozesse innerhalb einer Zelle zu studieren, entwickelt wurden, können sie auch auf Modelle, bestehend aus mehreren Zellen, und andere partikelbasierten Szenarien angewendet werden. Die Diffusion von Medikamenten in Tumoren, die Erkennung von Hohlräumen in Proteinen und die Darstellung von Molekulardynamikdaten (MD) aus Laserablationssimulationen sind nur einige Beispiele, die in dieser Arbeit angeführt werden.

CHAPTER 1

INTRODUCTION

For several years, systems biology has been an emerging, multi-disciplinary field combining biology with the quantitative fields of physics, chemistry, computer science, and engineering. Systems biology is closely related to biology itself and the biological main branches namely cell biology and molecular biology. There are, however, distinct points of view in each field or branch. Biology considers the structure and function of living organisms, their evolution, and their environment. The result is often a snapshot or static image of one organism where the structure and parts are clearly evident but the complex interactions are not reported. The scales covered in biology range from picometer (atoms) over micrometer (cells) to meters (organisms). Figure 1.1 depicts this wide range of scales and also defines the microscopic, the macroscopic, and the mesoscopic ranges. The boundaries between these three ranges are rather indistinct. Molecular biology, concerned with the function, interactions, and structures on the molecular or atomic scale, is considered with the microscopic scale since it can only be observed with the help of microscopes. The macroscopic scale on the other hand is visible to the naked eye. The mesoscopic range is in between the microscopic and macroscopic scales. On this scale, cellular biology takes place revealing the actual size, shape, and location of cellular components in tissues and colonies.

Cell biology, molecular biology, and biology all have in common that only one small portion of the whole scale is considered at one point in time, e.g. how does a mitochondrion look like. In contrast to that, systems biology investigates the characteristics and the complex interactions of all elements in a particular biological system [Palsson, 2000; Ideker et al., 2001]. In particular, systems biology investigates the regulatory mechanism that creates and maintains the observed structures and enables life. Thereby, several orders of magnitude in time and space have to be considered. Due to the large differences in space and time, i.e. femtoseconds and nanoseconds at the microscopic scale up to hours or days at the macroscopic scale, simplifications are necessary to be able to simulate such systems

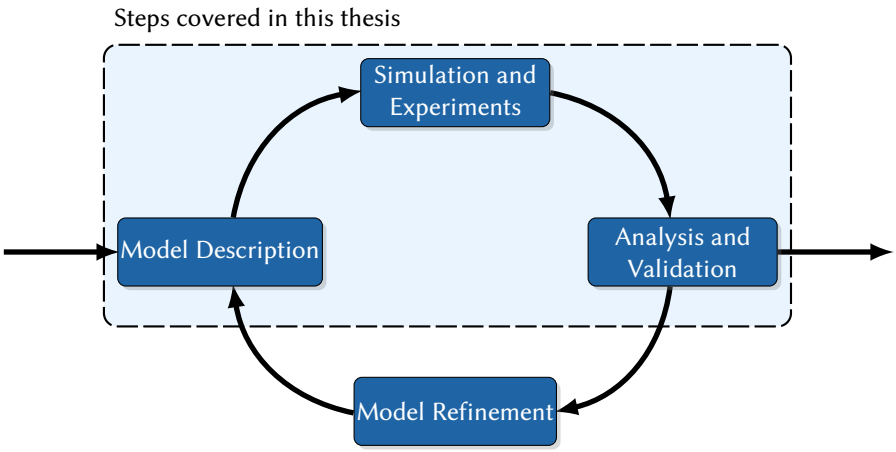


▲ **Figure 1.1 —** Biological objects differ several orders of magnitude in size covering the microscopic scale (pm to µm) over the mesoscopic scale (nm to mm) to the macroscopic scale (µm to m). The range covered in the visualizations in this thesis goes from picometer to millimeter. Hemoglobin and ribosome illustrations © David S. Goodsell and RCSB PDB [Goodsell, 2003, 2010].

today. A description of simulations within this context is given by Shillcock: "*Mesoscopic simulations retain only those molecular features that are believed to be relevant to the processes of interest*" [Shillcock, 2008]. In case of cellular mesoscopic simulations, the molecular dynamics of single proteins are neglected despite the fact that, e.g., protein folding or cleavage might be important for the interaction between intracellular proteins. The cleavage is not explicitly modeled and simulated but its known result, obtained from experiments or other simulations, is still incorporated and regarded in the simplified mesoscopic model.

Such mesoscopic simulations allow the investigation of effects that are difficult to study in experiments or for which experiments are unfeasible. Additionally, simulations are usually more cost effective than wet lab experiments and can be reproduced easily. Widely simulated biological processes include chemical interactions, metabolic pathways, gene regulatory networks, and cellular signaling pathways among others [Klipp et al., 2009]. In recent years, simulations utilizing the parallel hardware of current multi-core or many-core processors have become increasingly popular due to increases in computational power. The highly parallel architecture of modern graphics processing units (GPUs) is of particular interest because of the improved programmability and the low cost-performance ratio, due to the fact that GPUs are in fact off-the-shelf hardware.

Visual perception is responsible for the most part of the information a human perceives [Ware, 2004]. Visualization, therefore, has to play a crucial role in the process of data preparation and presentation. The data used for biological visualization originates from sensors in experiments or from simulations. The resulting data is often time-dependent and can contain several thousands of time steps covering several orders of magnitude



▲ **Figure 1.2 —** Model development typically is a cyclical process with four steps and can also be applied to design biological models. A theoretical model is built and simulated. The results have to be analyzed, compared with experimental data, and validated by domain experts. After refining the model, the next pass can take place.

in size, e.g. from individual proteins up to individual organs. This massive amount of data demands for tools that allow in-depth analysis and support data exploration to gain insight. For this task of visual analysis, scientific and information visualization techniques can be used. If the visualization tool supports the user during data exploration and reasoning, the process is also known as *visual data analysis* [Bertini and Lalanne, 2010]. In systems biology, associations play an important role and visualizations are a great way to make these visible. Visual data analysis can be used to interactively explore the data and to generate or confirm hypotheses.

1.1 Motivation and Scope

In systems biology, the characteristics and complex interactions of all elements in a particular biological system, like, e.g., the cell, are investigated using methods from systems theory to build mathematical models of processes within organisms. Model development can be split into four subsequent steps which are depicted in Figure 1.2. These four steps also match the course of action in biology. The process of developing a new model begins with experimental data and hypotheses or theoretical considerations. The model is then tested *in silico* on a computer with simulations and the model results are then compared with data obtained in *in vivo* or *in vitro* experiments. The results then have to be analyzed and validated by domain experts in the analysis step. This can be accomplished in different ways, but is mostly supported by appropriate visualizations like plots and diagrams or more advanced visual analytics methods as proposed in this thesis. Afterward, the results of the analysis can be used to validate the model and to adjust it for the next pass during model refinement. Subsequently, the refined model is used as a

new model configuration and the whole process is repeated. The complete cycle allows researchers to gain insight, to verify and improve the model under investigation, and to test new hypotheses.

The model of a biological cell was chosen as a model system for this thesis. Signal transduction processes in this cellular model are of special interest for both experts in systems biology and visualization. The dynamics of such processes are of interest for drug development and so far no 3D visualization approaches exist to depict this dynamic behavior. Within the process of signal transduction, external signals are transmitted by different mechanisms inside the cell toward their target. For instance, programmed cell death, or apoptosis, is initiated by a signal transduction making it highly relevant for cancer research. Designing drugs to specifically target only the signaling processes of cancerous cells can trigger cell death. Simulating such cellular systems and effects allows for a better understanding of those intracellular mechanisms and eventually leads to the development of more efficient drugs with fewer side effects. Other possible applications of cellular simulations include the simulation of actual experiments, e.g., for educational purposes, or toward the study of complex molecule interactions inside the cell. Cell-level simulations like the one proposed by Karr et al. [2012] might also complement wet lab experiments by reducing the expense of conducting extensive measurements and allowing for good predictions. In the long term, several cellular components or cells can be linked to build a dynamic and complex multiscale system. This system could then be used to study drug administration, e.g. in cancer research, on the organ level or in the whole organism [Enderling et al., 2009].

The aim of this work is to develop a mesoscopic simulation of selected intracellular and extracellular processes and visualizations, which are able to represent the simulation results in meaningful ways. In particular, *cellular signal transduction processes* will be studied. The effects of different transport modes inside the cell and the distribution of the signal are the focus of this research.

A prototypical application named *CellVis* was developed to combine the three stages of modeling, simulation, and analysis (see Figure 1.2, upper part) into one interactive tool. In this thesis, the last step, i.e. model refinement, is left entirely to the domain experts because it requires in-depth knowledge of the modeled system and its boundary conditions. It can, however, be argued that this step is inherently covered by the analysis step and through interactive graph editing where domain experts can adjust the model to achieve a model outcome consistent with experimental results. The *CellVis* system is intended to support the user in the overall process of model development. The simulation is performed either off-line or in lockstep with the analysis. For in-depth analysis, multiple coordinated views, including the three-dimensional model, tables, and plots, are provided. Coupling the simulation directly with the analysis allows the user to perform parameter changes on the fly.

1.2 Structure and Contribution

The structure of this thesis is as follows. Chapter 2 gives an overview of the biological fundamentals of cells and simulation techniques used in biology. The second part of this chapter deals with the basics of many-core architectures, including graphics hardware and its architecture traditionally dedicated to pure rendering. The chapter concludes with a description of visualization principles used throughout this thesis including a description of glyph-based rendering and volume visualization. These visualization principles build the foundation for the novel visualization approaches described in Chapter 5.

In the subsequent three chapters, the three building blocks of the model development cycle are described (cf. Figure 1.2). The modeling stage is detailed in Chapter 3 where the underlying structural model of the cell is explained in detail. In addition, a new graphical notation similar to the systems biology graphical notation (SBGN) is presented to allow for a visual representation of the interactions in the cellular model.

The second stage in model development, the simulation, is covered in Chapter 4. The simulation itself is based on the work of Michael Klann [2011] and relies on particles. This type of simulation is also known as agent-based simulation. The original simulation was closely inspected and adjusted to fit the highly parallel architecture of graphics hardware [Falk et al., 2011b]. Concepts for parallelization as well as optimizations are introduced to benefit from the potential performance gain of GPUs followed by an performance evaluation. Both simulations, the original one developed by Michael Klann and the parallelized GPU version were realized in sub-project A4 "*4D-spatial-temporal dynamics in cellular signal transduction processes: modeling, computational analysis and visualization*" of the Center Systems Biology (CSB) at the University of Stuttgart.

Chapter 5 begins with an overview of techniques that assist visual perception and data exploration. Afterward, interactive GPU-based visualization techniques are presented and applied to the field of systems biology supporting data analysis and validation. This includes methods enabling the analysis of particle data resulting from the simulation in both discrete and continuous ways [Falk et al., 2009, 2010b]. The atomistic visualization not only renders the particle data but also enhances the mesoscopic model by employing the internal structures of the proteins, depicting details down to the molecular level [Falk et al., 2012]. The newly developed microscopic visualization is designed to mimic the images obtained with confocal laser scanning microscopy (CLSM) in wet lab experiments [Falk et al., 2009]. Creating similar images facilitates the comparison between *in silico* data from simulations and *in vivo* or *in vitro* results from experiments. To obtain a continuous representation of the data, the density distribution of the signaling molecules can be computed either in object space [Falk et al., 2010b] or in image space [Falk et al., 2010a]. The same technique can be applied to approximate the surface of a protein in order to determine and track cavities within the protein [Krone et al., 2011]. At the end of the chapter, the proposed techniques are adjusted slightly for the visualization of membrane-bound receptor clustering [Falk et al., 2011a].

In Chapter 6, the individual components of modeling, simulation, and visualization are combined into the CellVis framework. The proposed framework covers these three steps in one single application guiding domain experts through the model development process. The prototype was developed in close collaboration with biologists with the aim of being usable by field experts that are not necessarily computer scientists. The software architecture of CellVis is highlighted and different application scenarios within CellVis are described and illustrated.

The results obtained in this thesis are generalized and discussed in Chapter 7. This chapter also discusses the application of simulation and visualization as tools for systems biology. Future directions of research are outlined in Chapter 8 including an perspective on multiscale models for simulating the effects of drug administration.

Besides the topics covered in this thesis, research has also been conducted in the fields of flow visualization and flow topology, real-time rendering, and cartography. Line integral convolution was applied to reveal flow features of steady flows [Falk and Weiskopf, 2008] and Lagrangian coherent structures were used to visualize the topology of unsteady flows [Falk et al., 2010c]. The work of rendering planets with atmospheric effects resulted from a collaboration with Tobias Schafhitzel [Schafhitzel et al., 2007]. Together with him, a tool for interactively generating cartographic panorama maps employing graphics hardware was developed in 2007 [Falk et al., 2007].

CHAPTER 2

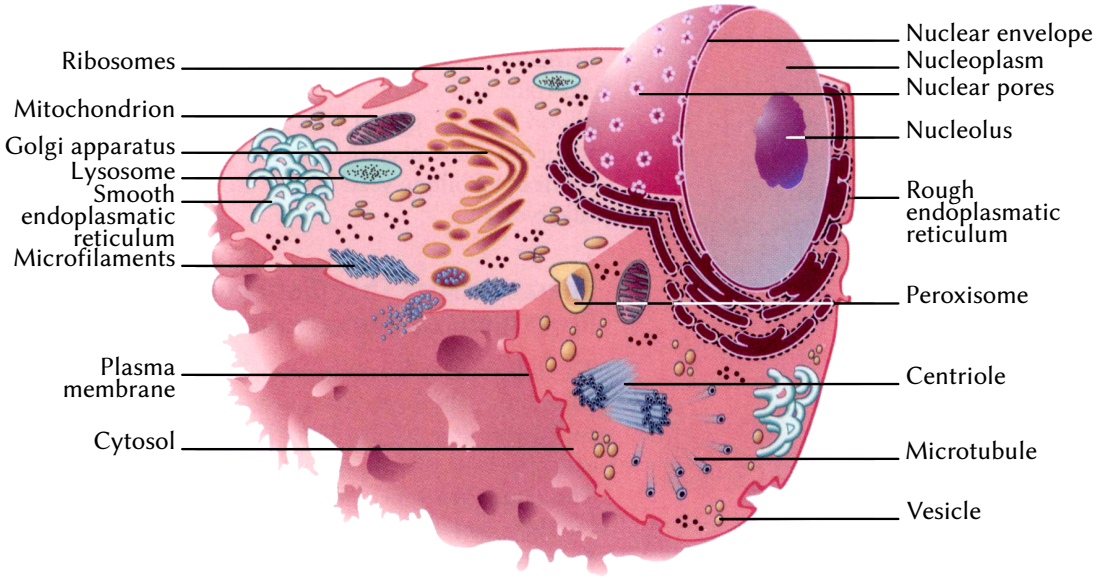
FUNDAMENTALS

This chapter first gives an overview of the cellular environment and simulation in biology. The second part introduces the concepts of many-core architectures and general purpose computation on graphics hardware. The chapter concludes with an exposition on principles of visualization that are used throughout this thesis, that includes the topics of particle visualization and volume rendering. Fundamental concepts of biology that appear in this chapter are adopted from Lodish et al. [2007] and Markl [2000] (English translation [Campbell, 1996]), if not otherwise noted.

2.1 The Cellular Environment

The human body consists of approximately 100 trillion (10^{14}) cells. There exists a multitude of different cell types in the human body like red blood cells, tissue cells, immune cells, liver cells, and so on. All cells have different functions but work together closely as an whole ensemble, thus facilitating life. Therefore one might agree with Lodish et al. [2007] that "*the cell is the fundamental unit of life*."

Cells can be classified into prokaryotic cells and eukaryotic cells due to their structural differences. The main difference is that organelles are not separated by membranes in prokaryotes. In this context, organelles (derived from organ) are defined as small distinct functional units. A prokaryotic cell can, therefore, be considered as a single closed compartment surrounded by a plasma membrane which is filled with cytosol. Bacteria are typical representatives of prokaryotes, i.e. single-celled organisms, like, e.g., cyanobacteria (blue-green algae) that can be unicellular or filamentous chains of cells. Organelles in eukaryotic cells are usually enclosed by lipid membranes, clearly separating the different parts in turn building small compartments.



▲ **Figure 2.1** — The structure of a eukaryotic cell. Organelles are not drawn to scale. Image: [Karp, 2010] © John Wiley & Sons, Inc.

The remainder of this section provides an overview of the eukaryotic compartments and processes taking place inside cells or in their vicinity. The processes described here focuses on the propagation of signals, i.e. signaling pathways, in the context of this thesis and less on the resulting effect like, e.g., cell division or cell death. The section concludes with a brief overview of various imaging techniques in biology, which are used to generate depictions of the interior of *in vitro* and *in vivo* cells with microscopes.

2.1.1 Compartments of Eukaryotes

The interior of eukaryotic cells is divided into several reaction spaces or compartments. Figure 2.1 depicts the inner structure of a generalized eukaryotic cell. The outer boundary of a cell is defined by the plasma membrane. The membrane itself is a 8 nm thick bilayer of phospholipid molecules, i.e. molecules with both hydrophilic and hydrophobic fronts. Proteins, including membrane receptors, are embedded between the phospholipids.

Inside the plasma membrane resides the nucleus. The space between the nucleus and the plasma membrane is called cytoplasm. It contains additional organelles, i.e. compartments enclosed by a lipid bilayer, and a liquid gel-like solution, the cytosol. The cytosol is comprised of water, cytoskeleton filaments, and other molecules and fills the space that is not enclosed by any other compartment.

The nucleus contains the genetic information stored in the DNA (deoxyribonucleic acid) and has an average diameter of 5 µm. The DNA stores all information coding necessary for the synthesis of proteins. The first step in the production of proteins is the process of *gene expression*, where genetic information is selectively read and used. This process is

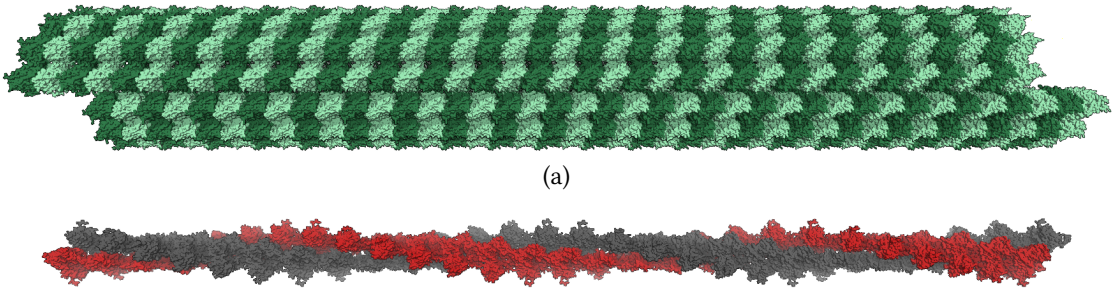
controlled at the level of transcription and produces mRNA (messenger ribonucleic acid) which is in turn transported to ribosomes. The nucleus itself is enclosed by the nuclear envelope, a double lipid layer which is 20 nm to 40 nm thick. The two layers of the nuclear envelope are fused together and form the nuclear pore complex (NPC). Nuclear pores are important for the import and export of molecules into the nucleus. Molecules smaller than 60 kDa might pass through the pores, whose opening diameter can be enlarged from 10 nm to 26 nm. The symbol Da stands for dalton where one dalton is defined as one twelfth of the mass of carbon-12 (^{12}C). Small molecules (< 5 kDa) diffuse through the opening. Active transport is used for larger molecules, taking about 2 min for the transport of a 17 kDa molecule.

The outer membrane of the nuclear envelope is connected to the endoplasmatic reticulum (ER). The ER can be divided into a rough part, studded with ribosomes, and a smooth part. It is responsible for production of membranes and the synthesis of fatty acids, lipids, and steroids. The ribosomes read mRNA and translate it to proteins. The products of the ER are finished, temporarily stored, and then shipped inside vesicles by the Golgi apparatus, another cellular compartment. The Golgi apparatus consists of a stack of membrane-enclosed cisternae. The finished products are placed into vesicles and then transported to their destination.

The energy needed in the cell is generated by further compartments, the mitochondria, by a process called oxidative phosphorylation that drives the formation of adenosine triphosphate (ATP) from adenosine diphosphate (ADP) and phosphate. ATP is split to ADP and phosphate by a multitude of processes when energy is needed. In addition to the aforementioned cellular compartments there exist also lysosomes, peroxysomes, and secretory vesicles in a cell. Secretory vesicles transport cell material to the plasma membrane to release it into the extracellular space. Peroxysomes process molecules using oxygen and lysosomes digest and recycle cell material and debris.

In the cytosol, various molecules coexist beside the water molecules including ATP, hormones, and different macromolecules. Typical macromolecules are polysaccharides (sugars), nucleic acids, and proteins. The structure and shape of a cell is maintained by a scaffold, the cytoskeleton. It is essential for cell locomotion with speeds up to 20 µm s^{-1} in the presence of chemical and stress gradients. The filaments of the cytoskeleton, i.e. its constituent parts, can be linked through the cell surface to the extracellular matrix and into a tissue. The filaments divide into three categories, the microtubules, the microfilaments, and the intermediate filaments.

The microtubules are built of the proteins α- and β-tubulin, which are arranged into a cylindrical form of thirteen columns (see Figure 2.2(a)). The outer diameter of the tubules is 25 nm with an open inner diameter of 15 nm. The main functions of microtubules are the stabilization and movement of the cell and the intracellular transport of proteins. Microtubules have a distinct polarity due to the combination of the two tubulin proteins. The negative end usually starts at the microtubule-organizing center (MTOC), also called centrosome, near the nucleus and growth happens at the positive end. The centrosome consists of two centrioles which are arranged perpendicular to each other and are composed of microtubules. The *motorized* or *active transport* of proteins along a microtubule



(a)

(b)

▲ **Figure 2.2 —** Structure of a microtubule and a microfilament. (a) α- and β-tubulin complexes, arranged in thirteen columns, form one turn of a microtubule. (b) One turn of an microfilament consists of 14 layers of 2 actin monomers each. For clarity the two strands are colored differently to highlight the pitch.

is handled by two specialized motor proteins. They convert the chemical energy of ATP into mechanical energy for movement. Dynein transports the cargo to the negative end of the microtubules, toward the MTOCs. The transport in the opposite direction from the center of the cell to the outer regions is carried out by kinesin.

The second type of cytoskeletal filaments, the microfilaments, are constructed of the protein actin, hence their alternative name actin filaments. Actin monomers bind at the tip of a growing filament to form a helical structure as depicted in Figure 2.2(b). The filaments stabilize the cell and can withstand both tensile and compressive forces. They are responsible for muscle contraction and movement of the cell. In case of chemotaxis, i.e. the directed movement to an attractant which might be a food source, the actin filaments elongate by polymerization at the leading edge of the cell. By extending toward the attractant and simultaneously degrading at the other end of the cell, the cell moves forward. Similar to microtubules, microfilaments are used for intracellular motorized transport. Here, different classes of the protein myosin are transporting their cargo along the microfilaments.

Intermediate filaments with a diameter of 8 nm to 12 nm can be grouped between microtubules and microfilaments (7 nm to 9 nm) in size. Around the nucleus, the intermediate filaments form a tightly connected network to hold the nucleus in place. The network extends to the plasma membrane, thereby serving as fixture for organelles in the cytoplasm.

In Table 2.1, the size and population of the main compartments and structures of a eukaryotic cell are depicted. The population numbers, given for a human liver cell, the hepatocyte, are only approximate because they are strongly dependent on cell state and type. Most eukaryotic cells vary in size between 10 µm and 100 µm, with some exceptions like zygotes. For comparison, viruses typically have a diameter of about 100 nm.

2.1.2 Intracellular and Extracellular Processes

Cells continuously monitor their surroundings and adjust themselves according to their environment. Intracellular and extracellular signaling is necessary for the cellular adjustment. This is expressed by the high number of proteins in cells dedicated to signaling processes. Between 10 % and 15 % of proteins in eukaryotes function as secreted extracellular signals, signal receptors, and signal-transducing proteins. The communication by signals employs pathways, which are cascaded reactions of proteins and hormones.

When an external signal activates a membrane-bound receptor and this receptor initiates an intracellular signaling pathway, the process is called signal transduction. Figure 2.3 illustrates the process of cellular signal transduction. Receptors located at the cell membrane are stimulated by an external signal and induce an intracellular signal. Subsequently, signal transduction takes place and the signal is transported through the cytoplasm by signaling molecules toward the nucleus. On its way, the signal can be amplified or deactivated in signaling cascades. Once the signal arrives at the nucleus, key molecules regulate gene expression in the nucleus as a response. This response can affect other genes, arrest the cell cycle, or even initiate ordered cell death. Feedback mechanisms from the nucleus can regulate the incoming signaling molecules in a positive or negative way.

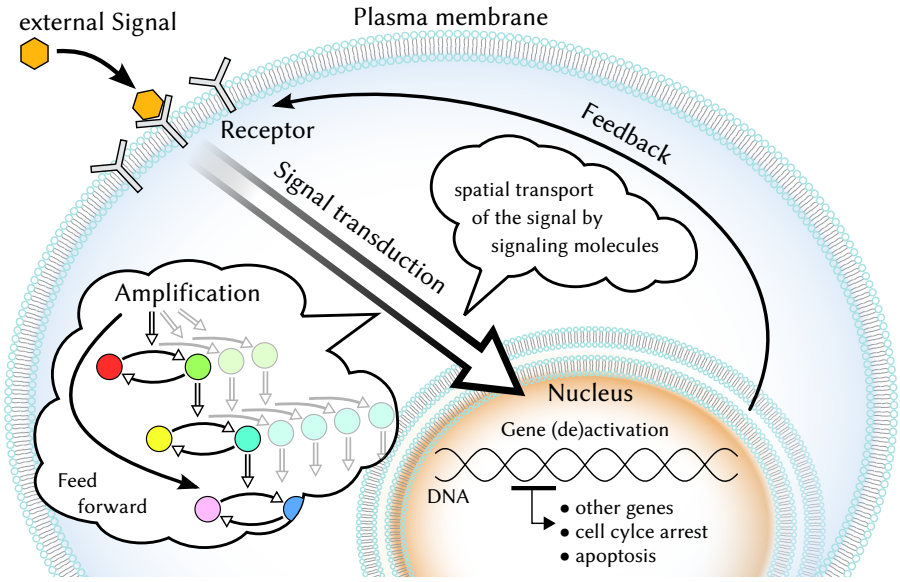
Besides signaling, a multitude of processes take place in a cell, like the cellular metabolism and movement, cell cycle and cell death, molecule transport, and DNA replication. The set

▼ **Table 2.1 —** Compartments and structures in a eukaryotic cell. Populations numbers denoted by [a] and [b] are given for a human liver cell.

Compartment	Size	Population
Plasma membrane	∅ 10 µm to 100 µm, 8 nm thick	1
Nucleus	∅ ≈ 5 µm	1
Nuclear envelope	20 nm to 40 nm thick	
Nuclear pores	∅ 100 nm to 120 nm	2000
Golgi apparatus	∅ ≈ 1 µm	up to 100 stacks
Mitochondria	∅ 0.5 µm to 1.5 µm	1000 to 2000 [a]
Cytoskeleton		
Microtubules	∅ 25 nm, 0.2 µm to 25 µm long	
Microfilaments	∅ 7 nm to 9 nm	5×10^8 actin molecules [a]
Intermediate filaments	∅ 8 nm to 12 nm	
Macromolecules		
Ribosomes	∅ 25 nm to 30 nm	
Proteins	∅ 1 nm to 100 nm	8×10^9 [b]
Water molecule	∅ 282 pm	80 % of cytosol

[a] in cuboid liver cell with 15 µm side length

[b] 20 % of cell mass, with 52.7 kDa average molecular protein weight in [a]



▲ **Figure 2.3 —** Cellular signal transduction. An extracellular signal is conveyed to the nucleus through conversion, amplification, and various modes of spatial transport. When the signal arrives at the nucleus, the gene expression is regulated as a response.

of cellular chemical reactions to maintain life in living organisms is called metabolism. Metabolism is responsible for growth and responding to environmental changes, reproduction, and preserving the cellular or the organism's structure. In cells, this includes harvesting energy, e.g., from sugars and constructing cellular components like proteins. Cell movement is induced by restructuring of the cytoskeletal filaments, e.g., during chemotaxis.

The cell cycle or reproduction cycle are of two kinds namely, *mitosis* and *meiosis*. *Mitosis*, i.e. asexual reproduction, in eukaryotes is highly regulated and is used for tissue replacement and regenerating damaged cells. During the cell cycle the DNA is copied and the cell is divided into two daughter cells with the aid of microtubules. The second kind, *meiosis*, is essentially sexual reproduction at the cellular level and is used solely for procreation.

If a cell is badly damaged or infected with a virus, it may die. This kind of cell death can occur in an unordered manner because of the structural decay. The damaged cells often release potentially toxic cell constituents which might damage other, surrounding cells. Cells might also die if they do not receive a life-maintaining signal or if they receive a death signal. In case the cell receives such a death signal, it disassembles itself in an orderly fashion. All its constituents are wrapped into vesicles and are thus no longer dangerous to the cell's surroundings. This process is called *apoptosis* or programmed cell death. *Apoptosis* is of importance in cancer research and during embryo genesis, where it is for example responsible for removing the webbing between fingers and toes in the embryo. The cells between the fingers disassemble themselves, leaving the individual five fingers unconnected.

Inside the cell, thousands of proteins and other constituents move through the cytosol. The movement might be undirected, i.e. affected only by diffusion. Directed transport takes place along microtubules and microfilaments (cf. Section 2.1.1). Another mode of intracellular movement is the transport of proteins and lipids in shipping containers, the vesicles. Vesicles are addressed with SNAREs (soluble N-ethylmaleimide-sensitive-factor attachment receptor), i.e. special proteins that only bind to specific target SNARE proteins located at the respective target compartment. Similar to single proteins, vesicles move by diffusion and motorized transport.

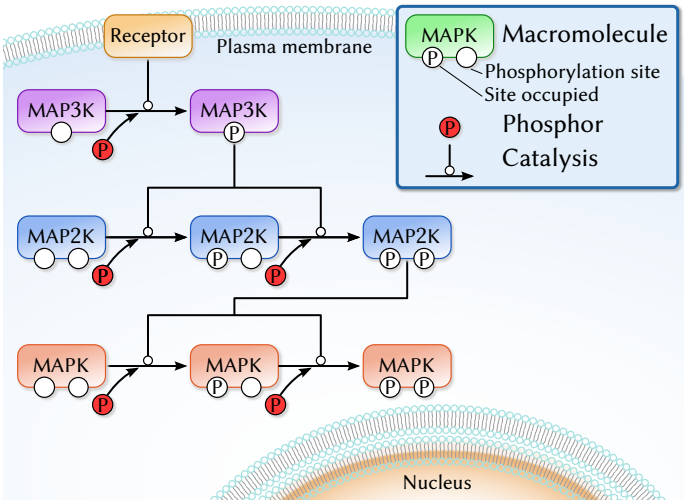
Two of the aforementioned processes are of special interest in the context of this thesis: mitogenic signaling and the transduction of apoptotic signals. Both, mitogenic and apoptotic signaling use intracellular signaling pathways to relay signals. In the course of this thesis, the generalized mitogen-activated protein kinase (MAPK) pathway is employed to simulate and visualize intracellular signal transduction processes. Subsequently, the MAPK pathway and the apoptotic pathways are explained in more detail.

Mitogen-Activated Protein Kinase Pathway

The mitogen-activated protein kinase (MAPK) pathway is a cascade of proteins which is used for cellular signal transduction, i.e. relaying an external signal over several tiers to the interior of cells. There exists a wide range of different MAPK pathways in mammalian cells and also in yeast cells [Kholodenko and Birtwistle, 2009]. In animal cells, the most prominent MAPK pathway is the extracellular-signal-regulated kinase (ERK) pathway, which is involved in cell growth, proliferation, and differentiation, often known. This pathway is also known as Ras-Raf-MEK-ERK pathway and is activated by growth factors. If this signal transduction pathway of the MAPK family is over-expressed or misregulated, the outcome can be cancerous diseases [Dhillon et al., 2007]. Other MAPK pathways (JNK and p38) are activated by stress, tumor necrosis factors (TNFs), or ultraviolet light and result in inflammation or apoptosis. Since the structure of these pathways is very similar, they can be summarized in a more general MAPK cascade. The scheme of the generalized MAPK cascade with three tiers is illustrated in Figure 2.4. In this thesis, this generalized pathway is used as a basic model for cellular signal transduction.

In the generalized MAPK cascade, the MAPKs are changed to an activated, double-phosphorylated form MAPKpp by the active form of MAPK kinase (MAP2K), i.e. MAP2Kpp. MAP2K itself is phosphorylated in two steps by MAP2K kinase (MAP3K). All phosphorylation reactions use the energy of ATP to initiate the activation and to bind the resulting phosphate group P_i. The active form of MAP3K, i.e. MAP3Kp, contains only one phosphate group and is in turn activated by membrane-bound receptors that are triggered by external signals, e.g. growth factors. This multi-component cascade transfers the external signal to an internal signal and allows amplification and regulation. The regulation of the signal includes the opponent deactivation of MAPKpp by phosphatases.

It is assumed that the upstream part of the cascade, i.e. MAP3K and MAP2K, is located in scaffolds at the plasma membrane and MAPK is the mobile component. Thus the signal



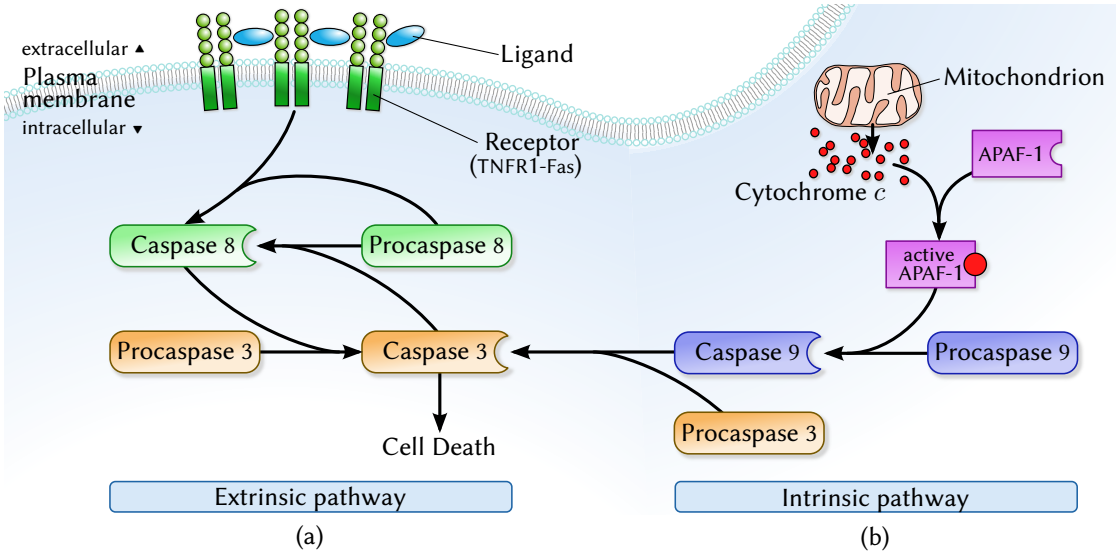
◀ **Figure 2.4** — Generalized MAPK pathway. The upstream part of the pathway near the membrane is activated by the receptor, i.e. phosphorylated to MAP3Kp. MAP3Kp then phosphorylates the next stage (MAP2K) twice which in turn double-phosphorylates MAPK, the final product of the pathway.

translocation to the nucleus depends on the arrival of MAPKpp, despite the dephosphorylation reaction everywhere else in the cell. The local excitation at the plasma membrane together with the global inhibition leads to spatial gradients; only a few MAPKp reach the nucleus. If motor proteins transport the MAPKpp along the cytoskeleton directly to the nucleus, the MAPK distribution in the cell will change and more molecules reach the nucleus [Kholodenko, 2003].

Apoptosis

Apoptosis is a significant physiological process since it enables the organism to remove unwanted or damaged debris of cells. The molecular mechanism of the apoptotic process is very complex so that the underlying network is difficult to understand in detail. However, two major pathways can be distinguished: the extrinsic and the intrinsic signaling pathway (see Figure 2.5). Both pathways have in common, that the activation of initiator caspases leads to activation of effector caspases. This process is known as the caspase cascade, the central and executing machinery of apoptosis. The caspase cascade is a network of specific proteases, the caspases. Proteases are proteins which are able to cleave and thereby activate other proteins. Extrinsic and intrinsic pathway both converge at caspase 3, which leads to cell death when activated.

The external stimulus of the extrinsic pro-apoptotic signaling pathway (Figure 2.5(a)) is the activation of procaspase 8 at the signal competent ligand-receptor clusters. These clusters on the plasma membrane consist of death receptors and the corresponding death receptor ligands. The ligand under consideration is the TNF ligand and the receptor belongs to the TNF receptor superfamily, e.g. TNF receptor of type 1 (TNFR1). In the following, a modified TNFR1 is considered, which consists of the extracellular domains of TNFR1 and the cytoplasmic part of the Fas receptor, a so called TNFR1-Fas chimera [Branschädel et al., 2010]. TNF exists as a homotrimer which is able to bind three TNF receptors. The TNFR1-Fas receptor contains four extracellular cysteine rich domains (CRD) of which the membrane distal part CRD1 enables the multimerization of TNFR1-Fas receptors [Branschädel et al.,

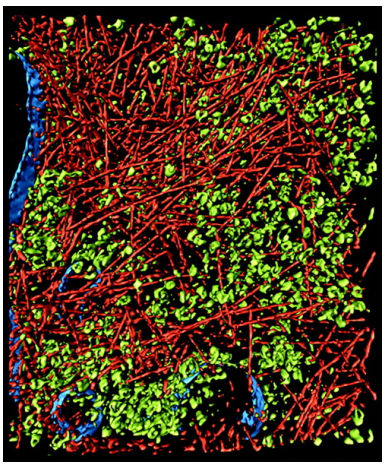


▲ **Figure 2.5** — Apoptotic pathways. (a) Extrinsic pro-apoptotic signaling pathway. Ligands bind to receptors (TNFR1-Fas) on the cellular membrane, which in turn triggers the signaling pathway. (b) Intrinsic mitochondrial apoptosis pathway. Cytochrome c is released by mitochondria and activates APAF-1. The active APAF-1 then initiates the caspase pathway.

2010]. The association of TNF ligands and TNFR1-Fas receptors occurs via the domains CRD2 and CRD3. The ligand-receptor clusters then activate the caspases located inside the plasma membrane upon cleavage and thereby stimulate the caspase cascade.

Two types of caspases are mainly involved in the reaction network of the extrinsic signaling pathway: the initiator caspase 8 and the effector caspase 3. Both caspase types exist in cells as proenzymes where the inactive form of caspase 8, i.e. procaspase 8, gets activated through cleavage, e.g., at death receptor complexes. The main part of the reaction network consists of the activation of procaspase 8 by caspase 3 and the activation of procaspase 3 by caspase 8. Finally, caspase 3 cleaves several proteins in the cell and thereby dismantles the cell leading to controlled cell death.

The second pro-apoptotic pathway is the intrinsic pathway (Figure 2.5(b)). This pathway is also known as mitochondrial pathway because it is mainly initiated by the mitochondria. The outer membrane of a mitochondrion releases proteins when it becomes permeable due to an apoptotic stimulus, like, e.g., Bcl-2 protein interactions. Among the released proteins is cytochrome c, which, once in the cytosol, facilitates the activation of caspases. The cytochrome c binds to the apoptotic protease activating factor-1 (APAF-1) and thereby activates it. The intermediate protein complex of APAF-1 and cytochrome c together with ATP is called apoptosome. It binds to procaspase 9, the initiator caspase of the mitochondrial pathway. Then the apoptosome cleaves the procaspase 9 to its active form, i.e. caspase 9. The caspase 9 then subsequently activates the effector procaspase 3, the final step of both, extrinsic and intrinsic apoptotic pathways. Once the cell enters the apoptotic state, its organelles degrade in an orderly fashion and the cell finally breaks apart into several vesicles.

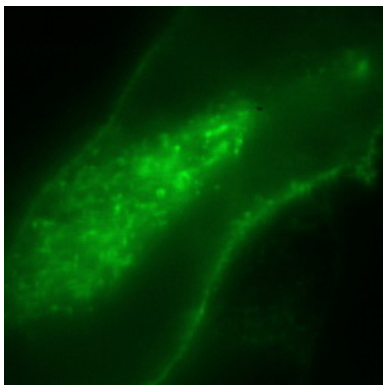


◀ **Figure 2.6 —** Intracellular structures obtained with cryo-electron tomography. Besides actin filaments (red), isosurfaces for membranes (blue) and ribosomes (green) are extracted from the 3D volume.
Image: [Medalia et al., 2002] © AAAS.

2.1.3 Imaging Cellular Compartments

Microscopes have to be used to obtain images of cells or cellular compartments due to the inherent, small scales ranging from nanometer to micrometer. Direct imaging of cells is possible with optical microscopes, also called light microscopes, down to approximately 400 nm. Higher resolutions can be obtained with electron microscopy (EM) where a focused electron beam is used instead of light. In transmission EM (resolution limit 0.05 nm), the beam is sent through a thin specimen slice. After transmission, the beam carries information about the internal structure. Surface information of specimen larger than 0.4 nm can be obtained with scanning EM (SEM). For a complete three-dimensional reconstruction of the specimen, like a cell, cryo-electron tomography can be used. The specimen is frozen instantaneously, thereby preserving the interior. Tomography is then used to generate a 3D volume of thin 2D slices. When the volumetric data is visualized, internal structures become visible with the aid of isosurfaces (cf. Section 2.5.3). Figure 2.6 shows an 815 nm × 870 nm × 97 nm section of a cell obtained with cryo-electron tomography and visualized by isosurfaces. Since EM does not allow for imaging live cells, its application to study dynamic cellular processes like signal transduction is rather limited and thus this microscopy technique is nonsignificant in the context of this thesis.

Optical microscopes, in contrast to electron microscopes, allow for imaging intact specimens and even living cells and are usually more cost-effective. Several microscopy imaging techniques have been established for this type of microscope. In bright field microscopy, the specimen is illuminated by a light source placed below and an image is obtained from the transmitted light. Since the illumination occurs through the specimen, most of the light is absorbed resulting in low contrast. The resolution is limited by the diffraction or Abbe Limit; single molecules are not visible. Dark field microscopy in contrast uses oblique illumination, i.e. only light scattered by the specimen is collected. The result is that the specimen appears brightly lit on a black background. The specimen might be directly imaged or after a preparation step like staining or metal coating (for SEM). With staining it is possible to emphasize specific parts of the specimen or even individual proteins. In order to stain proteins, the binding specificity of proteins to antibodies



▶ **Figure 2.7** — Fluorescent image obtained by confocal laser scanning microscopy (CLSM). TNF receptors are tagged with GFP, a fluorescent protein which is excited by the laser, thereby revealing their distribution and clustering.
Image: courtesy of Steffen Steinert © 2009.

is used. Purified antibodies are chemically linked to a fluorescent molecule and fed into the cell. Besides labeling of proteins with such fluorescent markers, it is also possible to indicate the state of signaling proteins [Allen et al., 2008]. The cell membrane is (partially) dissolved, so that the antibodies can drift in and attach themselves to the proteins, thereby marking them. Another option for staining is genetic engineering allowing for intact cells. When the modified gene is expressed, a hybrid protein consisting of the original cellular protein and a protein which fluoresces is produced. Exposing the cell to a light spectrum which is absorbed by the fluorophore, i.e. fluorescence microscopy, then reveals the location of the cellular protein, or to be more precise, the fluorescent part connected to it. The green fluorescent protein (GFP) is commonly used for fluorescence microscopy in cell biology. Other proteins fluoresce in colors including blue, cyan, yellow, orange, and red.

Fluorescence is also used in confocal laser scanning microscopy (CLSM), but the light source is replaced by a focused laser beam. As the name of CLSM suggests, an image is obtained by scanning the volume of the cell with the focal spot (with up to 1 fps, depending on the size of the cell or region of interest). The focus of the laser beam is very small (200 nm × 200 nm × 500 nm), which represents the current highest resolution in live cell imaging besides stimulated emission depletion microscopy (30 nm [cf. Schmidt et al., 2009]). The laser excites the fluorescent dye of tagged proteins. Emitted fluorescent light and reflected laser light are separated by a beam splitter. The fluorescent light is detected after passing a small aperture to improve the resolution. This aperture blocks almost all but a small portion of light around the focal point, leading to blurring. As the depth of field is very narrow, many samples have to be taken for a two-dimensional image or a volume. In Figure 2.7, an example of tagged TNF receptors is shown. The distribution and also clustering of the receptors is visible. In this example structural information about the cell membrane and the nucleus is inherently given by the fluorescence of the TNF receptors.

Multiphoton fluorescence microscopy (MFM) is another technique to image living tissue relying on the fluorescence of molecules. In contrast to CLSM, a laser with a wavelength in the infrared spectrum is used to excite fluorophores. Two or more low-energy photons are used to excite a fluorophore which then emits one single photon [Helmchen and Denk, 2005]. Due to the low energy of the photons longer observations times are possible and

slices of up to 1 mm thickness can be imaged. In addition, most of the background noise is suppressed because of the absorption of the excitation photons.

To overcome the diffraction limit of both CLSM and MFM, photoactivated localization microscopy (PALM) follows a different approach with photoswitchable fluorescent probes [Betzig et al., 2006]. Only a small number of molecules is excited for a short time in a single scanning pass. With the aid of the estimated point spread function, the coordinates of the individual fluorescent molecules are determined and the fluorophores are switched off afterward. By repeating the activation pass several times, a final image is generated by compositing the detected molecules at their respective positions. Samples being several micrometers thick can be imaged with PALM at an resolution of 30 nm × 30 nm × 75 nm [Juette et al., 2008].

2.2 Simulation in Biology

For a long time the only data source in biology were observations during wet lab experiments. With the advent of computers, it became possible to build models of biological systems that could be simulated. Such models are referred to as *in silico* models in order to separate them from *in vivo* (inside living organisms) and *in vitro* (inside test tubes) experiments. Today, especially in systems biology, computer-aided simulations are an important tool to explore and understand the processes of life or the intracellular mechanisms in particular. In addition, experiments might be partly or completely impossible to conduct, due to technical capabilities, or they might be very costly or time-consuming to perform. In such cases, computer-based simulations can help to fill the gap. Experiments are nonetheless vital nowadays to validate the *in silico* outcome.

As already mentioned, scales covered in biology range from picometer to meter requiring a wide range of simulation methods from molecular dynamics (MD) at the atomistic scale to population dynamics of organisms or agents in the system. Depending on the scale on which a biological model is investigated, several simulation techniques for the mathematical description of such processes exist [Turner et al., 2004; Takahashi et al., 2005; Tolle and Le Novère, 2006].

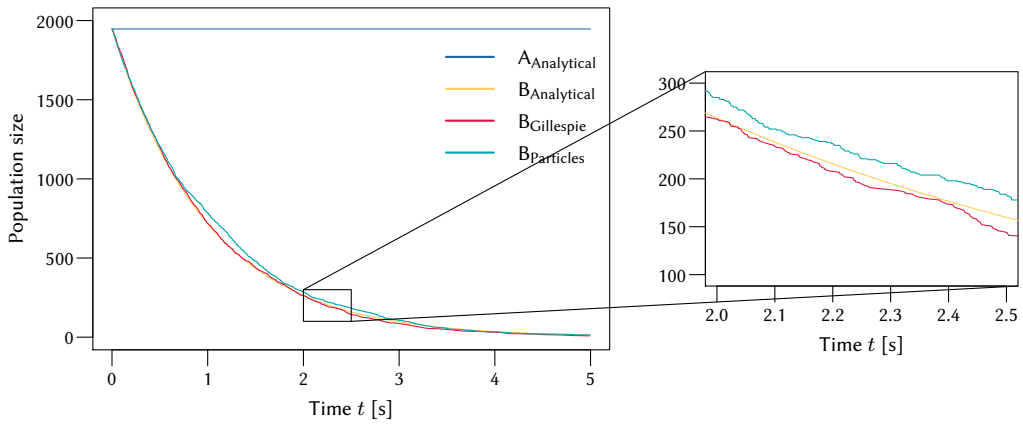
MD simulations cover atomic forces and resolve the folding or binding of proteins on nanometer and femtoseconds scale [Anderson et al., 2008]. However, to capture the spatio-temporal dynamics in cells significantly larger scales in time and space, i.e. on the macroscopic scale, are necessary. An overview of the methods that allow the modeling of the complete, mesoscopic cell is given by Takahashi et al. [2005]. Recently, Karr et al. [2012] presented an approach for simulating the life cycle of the human pathogen *Mycoplasma genitalium* in a whole-cell model. If large molecule numbers are assumed or the stochastic behavior of many processes is described, spatially unresolved models solely based on differential equations of the average concentrations are used [Takahashi et al., 2005; Pahle, 2009]. Stochastic aspects can be covered with the Gillespie method [Gillespie, 1977] or particle-based methods. Both, particle-based approaches and simulations with differential equations will be discussed in more detail in the subsequent sections.

A mixture between differential equations, stochastic approaches, and particles are voxel-based or grid-based methods where the reaction domain is discretized. They combine the three simulation types to benefit from their advantages. The price this comes at is that the overall computation is more costly than with differential equations but still not that precise as it would have been possible with particle-based approaches. However, the combined methods allow building simplified, voxelized models of cells. The simulation domain is divided into a grid for 2D simulations or voxels in 3D. Depending on the number of molecules allowed per cell, the 2D or 3D grid can be categorized into a microscopic lattice, where at most one molecule is allowed per cell, or a mesoscopic lattice with multiple molecules per cell. Typically, cellular automata [Weimar and Boon, 1994] are used in combination with these grids. Every molecule is represented by a cellular automaton with a finite number of states and is able to move from one grid cell to another depending on its state. Both microscopic and mesoscopic scales are supported. A major problem here is the explosion of molecular states because for each molecule and each grid cell a unique state is required. In the work recently published by Angermann et al. [2012], a voxel-based model is used in combination with a rule-based description of interactions. The shape of the outer membrane is adjusted during the simulation by means of an energy function enabling the interaction between two neighboring cells. Localized reactions, e.g. near the membrane, are adapted to the updated cell geometry accordingly because mutual feedback between cell shape and reactions seems to play an important role [Sneddon and Emonet, 2012].

2.2.1 Differential Equations

In systems biology, sets of ordinary differential equations (ODEs) are traditionally used to model signal transduction processes in a cell. A time-continuous model of the cell is constructed wherein spatial properties are neglected as reported by Takahashi et al. [2005]. While the most detailed approach assembles the signal transduction process from the individual protein with *in vivo* properties under *in vivo* conditions, the ODE framework describes the effective behavior and creates a continuous average throughout the simulation domain. Despite this constraint, differential equations are often used for parameter estimation in signaling networks, e.g. in cell populations [Waldherr et al., 2009]. Current central processing units (CPUs) and graphics hardware can be utilized to solve differential equations either analytically or numerically [Krüger and Westermann, 2005].

The laws of mass action kinetics [Waage and Guldberg, 1864] can be used to formulate chemical interactions with differential equations. The fundamental principle of mass action is found in the Michaelis-Menten kinetics [Michaelis and Menten, 1913], as well as Hill kinetics [Hill, 1910]. Both of them are of importance to the field of biology. Mass action kinetics defines the reaction rate to be proportional to the amount of reacting molecules. Two assumptions are, therefore, necessary. Firstly, molecules are uniformly distributed in space and secondly, the molecule population is large. The combination of both assumptions defines the simulation domain as a space where molecular concentrations can be assumed to be spatially constant. The assumption on large molecular



▲ **Figure 2.8 —** Analytical solution and the numerical solutions of the reaction A + B ⟶ A with the Gillespie algorithm and a 3D particle simulation (parameters are given in Appendix A.1). The closeup reveals the stochastic nature of both numerical methods.

populations, however, does not necessarily hold in biological systems. Often only a few molecules of a species are present leading to a low concentration and thereby leading to a poor approximation of the overall process. Despite this fact, ODE systems are still applied to biological pathways and networks because such systems are easy to set up and can be solved quickly and very efficiently.

Using mass action kinetics the reaction A + B ⟶ A taking place with a chemical reaction rate constant k can be expressed as a system of differential equations as follows

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} c_\mathrm{A} &= k c_\mathrm{A} c_\mathrm{B} - k c_\mathrm{A} c_\mathrm{B} = 0, \\ \frac{\mathrm{d}}{\mathrm{d}t} c_\mathrm{B} &= -k c_\mathrm{A} c_\mathrm{B}, \end{aligned} \tag{2.1}$$

where t is the time and c_X is the concentration of component X. The change in concentration for B is directly proportional to the reaction rate constant and the global concentrations of A and B. If one or both concentrations are zero, the reaction cannot take place anymore because one of the reactants is no longer available. The system in Equation 2.1 can be solved by the method of separation of variables and enforcing the initial condition $c_\mathrm{A} = c_\mathrm{B} = n$ yielding

$$\begin{aligned} c_\mathrm{A}(t) &= \text{const.} = n \\ c_\mathrm{B}(t) &= n \mathrm{e}^{-kt}. \end{aligned} \tag{2.2}$$

The analytical solution for an initial concentration of $c_\mathrm{A} = c_\mathrm{B} = 1 \times 10^{-7}\,\mathrm{mol\,L^{-1}}$, i.e. 1947 molecules for A and B, and $t \in [0; 5]$ is depicted in Figure 2.8. All necessary parameters can be found in Appendix A.1. However, if the reaction is slightly modified to A + B ⟶ C the solution is no longer trivial and has to be obtained numerically.

One of the basic assumptions when using mass action kinetics is the mandatory occurrence of many molecules as mentioned before. It has, however, been shown that a small number of molecules is, e.g., sufficient to initiate the apoptotic process. In the example of the MAPK pathway, the activation of the first stage is localized near the plasma membrane. If an ODE system is used to model this pathway, localization is lost and the first stage of the MAPK is uniformly distributed in the cytoplasm.

Partial differential equations (PDEs) can be used to deal with this specific problem. Here, the simulation domain is divided into several sub-compartments or sub-volumes. When using small sub-volumes, finite element methods can be used to evaluate the system of PDEs. Compartmentalization, i.e. a subdivision following the boundaries of compartments, allows, e.g., to distinguish between plasma membrane, nucleus, and other components of the intracellular signaling pathway [Kholodenko, 2003]. Inside the compartments, the concentrations are again considered to be constant but they might differ between different compartments allowing for localized effects to become visible. Exchange of molecules between compartments, i.e. adjustment of the local concentrations, can also be modeled by differential equations. Defining transport rates between different compartments allows for the formulation of the whole system with ODEs.

The system being modeled with PDEs is much closer to the actual cell than with ODEs but still does not consider the intracellular architecture as the different compartments are just sub-volumes of the overall simulation domain. Filaments of the cytoskeleton or other cellular compartments are difficult to model with PDEs. Motorized transport along cytoskeletal filaments could be simulated by defining a special compartment for all molecules currently in transport. Since there is no spatial relationship given other than that a certain number of molecules is actively transported, there is no further information on where the molecules are transported to. A situation that is implicitly and best handled by particle-based simulations.

Another issue concerning both ODE and PDE systems is that everything is continuous particularly with regard to chemical reactions. Molecules, however, are discrete as well as reactions, which are modeled to be discrete events in time. When the numbers of molecules are considered to be discrete and the reaction rates are expressed as probabilities the resulting system of stochastic differential equations (SDEs) has the form of a chemical master equation. The stochastic simulation algorithm (SSA) of Gillespie [1977] can be used to solve the SDE by a Monte Carlo (MC) approach (see [Gillespie, 2007] for an overview of stochastic simulations). Random numbers are used to determine how much time has elapsed between two steps and which reaction took place. In the obtained time span the chosen reaction takes place exactly once and the molecule numbers are adjusted according to this reaction. The updated molecule counts are then used as input for the next iteration. Figure 2.8 shows the result of the SSA for the example reaction $A + B \longrightarrow A$ in comparison to the analytical solution and the results of a particle simulation. The deviation from the analytical solution is due to the stochastic nature of the process which is clearly visible on the right. Given a high number of different realizations, i.e. executing several simulation runs with different random seeds, the averaged results will be identical to the analytic result. The SSA is perfectly suited for parallelization on current graphics

hardware (cf. Section 2.4). Several parallelized simulation runs can be executed in parallel allowing for statistically relevant data [Li and Petzold, 2010].

One of the advantages of Gillespie's algorithm is that it is event-based, i.e. the step width depends on events and not on discrete time steps. The drawback of possibly small step widths is eliminated with the use of the approximate τ-leap algorithm [Gillespie and Petzold, 2003] where several time steps are skipped in a way that the results remain similar. The computation of the time span τ is more complex compared to that in the original SSA, but the large time steps result in fewer simulation steps. In contrast to the SSA, the net change of all molecules during the period of τ is considered. All reactions take place using reaction probabilities, converted from the chemical reaction rates, and the number of molecules is adjusted for this time span.

In all these methods, spatial aspects are not considered since the populations are still uniformly distributed. Marquez-Lago and Burrage [2007] improved the τ-leap algorithm by introducing several sub-volumes similar to the PDE-based approaches. The so-called *next sub-volume method* is employed to realize the molecule transfer between different volumes. The next sub-volume methods extends the chemical master equation to a lattice-based reaction diffusion master equation [Stundzia and Lumsden, 1996; Elf et al., 2003]. Inside a single sub-volume the distribution of molecules is once again assumed to be uniform. The spatial subdivision of the simulation domain enforces the second assumption of mass action kinetics, i.e. molecule numbers have to be high. Concentrations might differ between different sub-volumes leading to localized peaks or intracellular gradients [Kholodenko, 2003]. In this model and also in the PDE-based model the cell is, however, only roughly approximated and cellular components are ignored. This does not correspond to the situation in real cells where the intracellular space is packed with various cellular compartments, the filaments from the cytoskeleton, and other molecules and proteins (cf. Section 2.1.1). It has also been shown by Ellis [2001] that the molecular crowding should not be neglected.

Existing frameworks like CellDesigner [Funahashi et al., 2008] or TinkerCell [Chandran et al., 2009] often rely on ODE solvers for simulating their networks. Karr et al. [2012] also apply ODE solvers to simulate their whole-cell model. This is due to the fact, that simulating ODEs usually takes less time than simulating particles by neglecting spatial effects.

2.2.2 Particle-based Methods

To simulate crowding effects, non-uniform concentrations, and low numbers of molecules, the population model with differential equations has to be exchanged with a more precise one, especially in the context of signaling pathways. Particle simulations with discrete particles moving individually in time and space are particularly suited to perform these tasks.

Particle-based simulations have been used and studied for a very long time. Smoothed particle hydrodynamics (SPH) introduced by Gingold and Monaghan [1977] or MD simulations introduced by McCammon et al. [1977] are popular. Recent works on SPH cover cohesion effects of fluids [Alduán and Otaduy, 2011] as well as granular media [Schechter

and Bridson, 2012]. In recent years, the calculation power of graphics hardware was harvested for these techniques. Kipfer et al. [2004] presented a GPU-based system for particle simulations, which was, however, primarily intended for rendering and animation and not scientific simulation. Van Meel et al. [2007] use the modern general purpose GPU framework CUDA (cf. Section 2.4.3) for a fully functional molecular dynamics simulation on graphics hardware.

In contrast to such physically motivated simulations, which usually simulate a large number of particles with rather simple particle interactions like pair potentials, there exist simulations taking on a different approach. For example in systems biology the movement of molecules and proteins inside a cell is simulated, where the internal structure of the cell—the cytoskeleton—is used to direct the protein movement toward the nucleus by motor transport [Klann et al., 2009], resulting in a simulation of less particles but with rather complex behavior. MD simulations cannot be used for this type of scenario due to the high computational effort of such interactions. Only short periods in the time of nanoseconds and spatial range of picometer can be simulated, which is not sufficient for biological systems where processes take up to several seconds, minutes, or even hours. However, these methods are well suited for sub-cellular problems like protein folding [Anderson et al., 2008] or receptor binding. To allow for larger time scales, interactions in a cell have to be handled on a coarser, mesoscopic level where each molecule or protein of interest is represented by a single particle with a discrete position.

The Gillespie approach mentioned in the previous section can be seen as a model which combines continuous differential equations and single particle methods because reactions occur only on a per molecule-basis once per time step. Inside the sub-volumes though, the molecules are not considered to be individualistic and, hence, crowding effects cannot be covered. Crowding effects are best simulated with Brownian dynamics (BD), where random forces are used to model interactions between diffusing molecules. The BD simulation solves an n-body problem of n interacting molecules in a stochastic manner. Since the simulation approach is continuous in space and time, arbitrary time steps can be used. The molecule propagation follows the Langevin equation and the forces used to model the diffusion of molecules are obtained by considering all other molecules resulting in an algorithmic complexity of $\mathcal{O}(n^2)$.

In order to reduce complexity of BD, the n-body problem can be reduced into a two-body problem where Green's function is applied [van Zon and Ten Wolde, 2005]. The basic idea on which Green's function reaction dynamics or GFRD relies is the maximum possible movement between two time-steps. In the event-driven, stochastic simulation the maximum movement for each particle in which no collision occurs between two single particles is computed at every time step. Thereby, the shortest time span can be determined as step width of the current iteration. The adaptive step width is clearly an advantage over BD but only for low molecule concentrations. If the molecule concentration increases, the number of collisions will increase too and, hence, the step width will shrink and the benefit over BD is reduced. In addition, it is very difficult to parallelize the event-driven simulation to profit from the performance given by today's many-core architectures (see next section), e.g., to simulate larger systems.

In particle-based simulations with chemical interactions, a reaction between two colliding molecules is similar to the coagulation of particles in the model of von Smoluchowski [1917] only for elastic assumptions and also depends on their ability to form dimers and polymers. Two molecules or particles might coagulate or react with each other if they are very close to each other. The reaction space is defined by a sphere around one of both particles. If the second particle enters this reaction space, the reaction might happen. The radius of the sphere, i.e. the binding radius, is determined depending on the nature of the reaction, the reaction rate, and other criteria. In a similar way, two particles might separate from each other if one leaves the space given by an unbinding radius. The simulation itself uses discrete and uniform time steps. Care has to be taken in the choice of the time step. Reaction events might be missed if the time steps are too large because molecules might cross whole reaction spaces within a single movement. For certain PDEs, however, the Courant–Friedrichs–Lewy condition can be used to determine the step length in space and time. The Smoluchowski model has been applied to cellular systems by Andrews et al. [2010] with the Smoldyn tool (from Smoluchowski dynamics) and also to chemical reaction diffusion systems [Dematté and Mazza, 2008]. Popular simulation frameworks using particles besides Smoldyn include MCell [Stiles et al., 1996; Casanova et al., 2004], ChemCell [Plimpton and Slepoy, 2005], and also GPU implementations for reaction diffusion systems [Dematté, 2010]. Further particle-based methods applied to biological systems can, e.g., be found in the overview given by Tolle and Le Novère [2006].

With BD, Green's function, or the Smoluchowski model it is possible to simulate the diffusion processes and chemical reactions in different environments. The internal structure of the cell though is not entirely taken into account and there is hardly any interaction between molecules and cellular structures. The works of Lapin et al. [2007] and Klann et al. [2009] also employ the model of Smoluchowski but focus more on the intracellular architecture and interactions between molecules and structural elements like, e.g., the cytoskeletal filaments. The interactions are used to model state transitions of the molecule's movement. In this way, it is possible that a molecule, diffusing through the cell, collides with a filament and then gets transported by motor proteins.

The parallelized particle simulation used in this work is based on the simulation of Klann [2011] and is described in more detail in Chapter 4. Smoluchowski dynamics [von Smoluchowski, 1917] are used to model the molecular interactions and molecules are treated as spheres. The proxy spheres enclose the molecular structures with their radii set to the hydrodynamic radii of the physical molecules. Short-range collisions between particles are used to model chemical reactions. Interaction rules between the individual molecules are based on biochemical reaction rates [Pogson et al., 2006]. In the following, the targeted platform for the parallel simulation, i.e. today's many-core architectures, are briefly discussed.

2.3 Many-core Architectures

Computer architectures can be classified into four classes as described first by Flynn [1972]. Depending on the number of instruction and data streams four classes emerge: SISD (single instruction, single data); SIMD (single instruction, multiple data); MISD (multiple instruction, single data); and MIMD (multiple instruction, multiple data). Nowadays, the term SIMD is often replaced for GPUs by SIMT (single instruction, multiple threads) or by SIMS (single instruction, multiple streams) given their underlying hardware architecture, which is a streaming multiprocessor, and their programming models (cf. Section 2.4).

Many-core architectures can be classified either as a SIMD or a MIMD model depending on the level of independence of the single cores. They have all in common the feature that the computing unit consists of several (many) interconnected processing cores sharing the same interface to the main memory, e.g. a single workstation and no network of distributed systems.

CPUs found in recent consumer offerings match the many-core architecture type and are called *multi-core CPUs*, consisting usually of less than eight cores. Another component of those computers is also among the many-core architectures: the GPU. Their underlying architectures, however, differ in several aspects as will be shown in the following sections. A direct performance comparison between these two architectures is difficult and depends on the given task (cf. Section 4.4). For a rough comparison, the theoretical maximum number of floating point operations per second (flops) and the maximum memory bandwidth is given for both, CPU and GPU. The fastest CPU with six cores (Intel Core i7 3960X), at the time of writing, has a peak performance of 177 Gflops in double precision computations and a memory bandwidth of 51.2 GiB s^{-1}. The NVIDIA GeForce GTX 580 (GPU) performs with 198 Gflops (in double precision) whereas the ATI Radeon HD 7970 (GPU) achieves 947 Gflops. However, in single precision the GPUs perform at much higher rates between 1.58 Tflops (NVIDIA GeForce) and 3.79 Tflops (ATI Radeon). Because of the efficient hardware layout, the memory bandwidth in GPUs is also higher than on the CPU, i.e. 327.7 GiB s^{-1} (NVIDIA GeForce) and 264 GiB s^{-1} (ATI Radeon).

2.3.1 Multi-core CPUs

CPUs are probably the most commonly known type of multi-core central processing units but there are other types. Systems on a chip (SoC) or multiprocessor systems on a chip (MPSoC) integrate all components of a specific system into one integrated circuit. This might be a radio receiver combined with an amplifier or even a computer. Another example is found in digital signal processing. Here, DSPs (digital signal processors) are used for time-critical computations in many fields with a specialized instruction set.

In 2008, Intel proposed Larrabee, a new multi-core architecture based on x86 dedicated to both central processing and graphics processing [Seiler et al., 2008]. The project was canceled in 2009 and the main parts of this concept, excluding the graphics processing part, were merged into the Intel *many integrated core* architecture (Intel MIC) which is used as a co-processor in several high performance computing platforms.

Different programming models exploiting the parallel SIMD or MIMD architecture of multi-core CPUs are available in several programming languages. One solution for the parallelization of code for multiple cores is OpenMP.[1] The code (Fortran or C/C++) is instrumented with additional compiler directives allowing for a fine-granular parallelization and is used in this thesis for optimization of the CPU-side code. Threads allow for a more coarse-grained parallelism, e.g. for independent computational tasks in the same process. If the application is partitioned into several parallel processes the available *shared memory* can be used for inter-process communication. Other alternatives include but are not limited to the message passing interface (MPI) and sockets, which are both easily expandable to networked systems and interconnected clusters of computers.

2.3.2 Graphics Processing Units

In recent years, the hardware of graphics processing units (GPUs) has changed dramatically from simple graphical output buffers over specialized processors for rendering and displaying 3D scenes and images to a many-core architecture for various applications. More and more parts of this fixed function pipeline were replaced by user-programmable components with new added functionality (cf. Section 2.4.1 for details).

In modern GPUs, more units are dedicated to computing and the memory interconnection is faster compared to CPUs. GPUs also include more cores but of those which are specialized. NVIDIA GPUs are partitioned into 16 streaming multiprocessors, where each multiprocessor consists of 32 identical cores (data of NVIDIA GeForce GTX 580). This concept can be classified as SIMT model and offers high data parallelization because the cores of one streaming processor perform the same set of instructions on different data.

2.4 General Purpose Computation on GPUs

Despite being driven mainly by the gaming industry, current graphics hardware provides a functionality, which is of special interest to the scientific community. The increased generalization and computational power of GPUs led to a more general application of GPUs to fields other than computer games, like, e.g., video encoding and decoding, image filtering, acceleration of graphical user interfaces, and website rendering with WebGL. This class of GPU usage is known as general purpose computing on graphics processing units or GPGPU and has been an active research area in the last decade.[2] The different programming models and their constraints for GPGPU programming are described in this section.

The major aim of GPGPU is to make use of the parallelism available in GPUs. The many-core architecture of a GPU with the streaming processors handles the incoming data in a stream-like manner (*data stream*). The computation called *kernel* is executed

[1] http://www.openmp.org
[2] http://www.gpgpu.org

on each core of the streaming processors in parallel and operates on the data stream. Two programming paradigms for GPGPU coexist in practice. The first one employs the rendering pipeline of the GPU (cf. Section 2.4.1) for the computation of non-graphic applications. Programmable units, called *shaders*, represent *kernels* working on the data stored in graphics memory. The second approach is more programmatical and hardware-oriented and hides the graphic functionality from the user. At the time of writing, three major frameworks exist for this approach. DirectCompute from Microsoft is part of DirectX[3] and is only available for Microsoft Windows. In contrast, NVIDIA's Compute Unified Device Architecture (CUDA) [NVI, 2011] is platform independent but requires a GPU from NVIDIA. The third and youngest framework is the open compute language (OpenCL) [Khr, 2011]. It is available for multiple platforms and operating systems and different types of many-core processors, i.e. AMD GPUs, NVIDIA GPUs, and x86 CPUs.

In the course of this thesis, CUDA is used for the simulation on GPUs because OpenCL was not available at the time of development. However, NVIDIA CUDA code can be transformed into OpenCL code with minor modifications [Karimi et al., 2010]. The advantage of using CUDA is that the programmatic and runtime overhead is small compared to that of OpenCL code [Karimi et al., 2010; Du et al., 2012]. In the following sections, details are provided on the rendering pipeline, on which most visualizations in this work are based on, followed by descriptions of general purpose computing with shaders and CUDA.

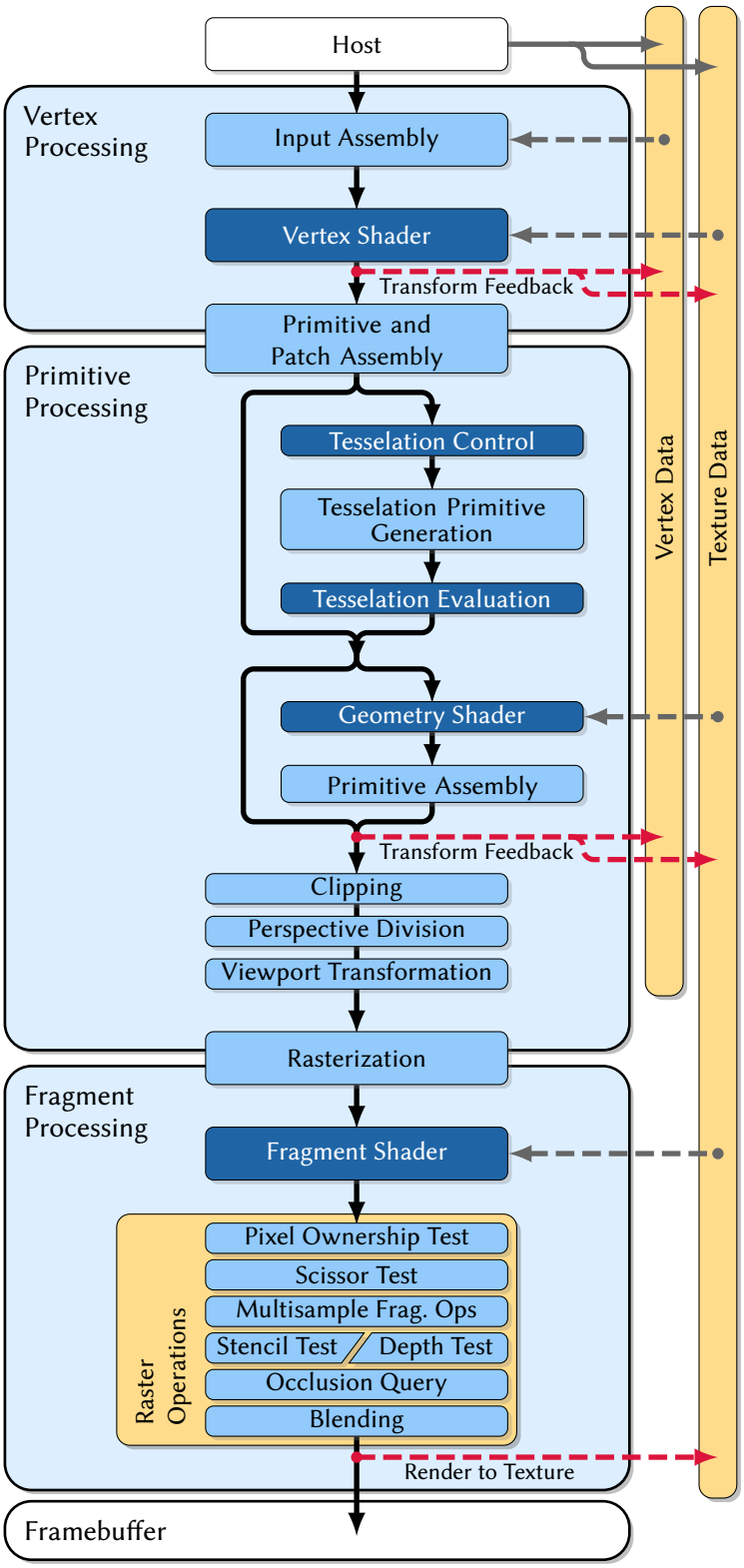
2.4.1 Rendering Pipeline

When performing the primary function, the polygonal rendering of a 3D scene, the GPU projects the three-dimensional scene onto a 2D image. This mapping is performed over several steps following the *rendering pipeline*. The OpenGL rendering pipeline of current GPUs, depicted in Figure 2.9 and described by the OpenGL specification [Segal and Akeley, 2011], is optimized to perform this task.

The rendering is initiated by the host side (CPU) with a draw call of primitives, which can be points, lines, or triangles. These primitives consist of one, two, or three vertices and their respective connectivity. A vertex holds its position and optional data like color, normal information, texture coordinates, and other generic data called *vertex attributes*. The vertex data can be stored in both main and GPU memory depending on the requirements of the current task. In addition to organizing the vertex for suitable representation, the host also initializes the texture data in GPU memory. Textures can either be one-dimensional, two-dimensional, or three-dimensional. Each texture is a data array whose basic data elements are called texels (1D and 2D textures) or voxels for 3D textures holding up to four scalar values. The texture data is accessible at various stages throughout the rendering pipeline. The host-defined data is processed by the GPU in a stream-like manner in the three main stages of the rendering pipeline:

Vertex Processing: After the draw call for a primitive is initiated by the host, the vertex data is processed by the *vertex processing* units. As the name suggests, the data is

[3] http://msdn.com/directx



▲ **Figure 2.9** — OpenGL 4.0 rendering pipeline according to the OpenGL specification of Segal and Akeley [2011].

handled on a per-vertex basis and the connectivity information given by the draw call is temporarily neglected. The first step is the *input assembly* where the vertex positions and attributes are fetched either directly from the input stream and the stored vertex data or by using indices. The programmable *vertex shader* processes the vertex and modifies its attributes. The vertex coordinates themselves are transformed from object space coordinates to clip coordinates using affine transformations. This transformation can be decomposed into three parts. The model transformation first maps the object coordinates of the vertex into world space followed by the camera transformation resulting in camera coordinates. The projection of the camera coordinates to non-normalized homogeneous clip coordinates is the last step. The predefined vertex attributes other than the vertex position might also be transformed and altered or new attributes may be created in the vertex shader. Together with the connectivity information, the vertices are put together to render primitives or patches in the last step, the *primitive and patch assembly*, which hands the processed data over to the next stage.

Primitive Processing: In the *primitive processing* stage, the primitives coming from *vertex processing* can be further refined in the two optional process paths *tesselation* and *geometry processing*. The resulting render primitives are then clipped against the viewing frustum. Primitives which intersect with the viewing frustum are cut at the intersection and the outside part is discarded. The part remaining inside the frustum is then re-retesselated into triangles. If back face culling is enabled, all render primitives facing away from the camera are also discarded. After clipping, the homogeneous clip coordinates of the vertices are normalized by the *perspective division* resulting in normalized device coordinates. The *viewport transformation* maps the device coordinates into window coordinates, the actual rendering viewport. The remaining render primitives pass the *rasterization* where they are rasterized according to the target image and fixed to specific pixels in image space. At each raster position, the vertex attributes, including the window coordinates, are interpolated from the render primitives and are stored in a single *fragment*, i.e. a pixel (color and position) with additional information like, e.g., depth and opacity. A special feature of the rasterization stage is the *early z-test*. If the current fragment has a greater depth than a previously rendered fragment, the computations the current fragment can be aborted prematurely. However, the *early z-test* is automatically disabled if the subsequent *fragment shader* manipulates the fragment depth or uses `discard` to reject fragments.

The tesselation or subdivision of primitives works solely on patches, which are a special OpenGL primitive type. In the programmable *tesselation control shader*, the subdivision scheme, i.e. isolines, triangles, or quads, is set as well as the number of subdivisions per edge and face. The actual subdivision of a primitive patch is performed by the *tesselation primitive generation* that produces common primitives. The resulting primitives are then processed in the programmable *tesselation evaluation* shader. Here, working in local patch coordinates, the shader might manipulate the vertex positions of the generated primitives.

The primitives coming either directly from the vertex processing or the tesselation can be modified in the *geometry processing* unit. The programmable *geometry shader* works

on a per-primitive basis and can create or discard vertices for extending the underlying primitive, e.g. transform a single point to a hexagon. The resulting modified or newly created vertices are then re-assembled to render primitives by a second *primitive assembly*. As mentioned before, the tesselation of patches and the subsequent geometry processing are both optional and are skipped when not needed.

Fragment Processing: The last step in the rendering pipeline is *fragment processing* where the fragments of the rasterized primitives are handled before they are stored into the *framebuffer*. The most important part of this stage is the programmable *fragment shader*. The fragment shader accesses the fragment attributes set by the rasterization and uses them for various computations, e.g. accessing textures. Common tasks include, but are not limited to, per-pixel illumination, depth adjustment of the fragment, or discarding a fragment based on its alpha value. When the fragment has passed the fragment shader, several raster operations are performed. The first operation is the *pixel ownership test*. Here the pixel at the fragments location is checked if it is owned by the OpenGL context or not. In case it does not belong to the current OpenGL context, the underlying window system decides whether to accept the fragment or discard it, e.g., if the window is obscured. The next test is the *scissor test*, where the fragment is only accepted when lying inside a rectangular mask. A coverage mask from multiple samples per pixel is computed by the *multisample fragment operations* and then used to modify the alpha value of the fragment. *Stencil test* and *depth test* are combined tests, where the depth test discards or accepts the fragment based on its depth value and the content of the depth buffer at the position of the fragment. For the *stencil test* a special buffer, the stencil buffer, is used as an additional mask to accept fragments by counting fragments passing or being rejected by the depth test. The number of fragments passing the depth test is determined by the *occlusion query* and can be reused for optimization of subsequent rendering passes. When a fragment has passed all previous tests, it can be written to the framebuffer or alternatively to a texture (*render to texture*). The last raster operation, i.e. *blending*, determines how the fragment is combined with the previously stored color of the framebuffer. If disabled, the fragment color simply replaces the stored color. Otherwise the two colors, the fragment color and the framebuffer color, are interpolated depending on the employed blending function. When multisampling is enabled, all raster operations are performed for each fragment sample instead of only once per fragment. The tests are only performed for samples with a coverage value greater than zero.

The vertex, tesselation, geometry, and fragment shaders can be programmed with the high level languages GLSL (OpenGL shading language) [Kessenich et al., 2011] and CG for OpenGL or their DirectX counterpart HLSL (high level shading language). Some of the other steps, like clipping and the raster operations, are dependent on global GPU states configurable by the user. Additional information on these states and the rendering pipeline can be found in the OpenGL programming guide [Shreiner and the Khronos OpenGL ARB Working Group, 2009] and the OpenGL specification [Segal and Akeley, 2011].

Intermediate results, including vertex positions and vertex attributes, can be written to memory by using the *transform feedback* mechanism. This allows, e.g., to reuse the results of one pass as input of another one. The *render to texture* mode for pixel data

is similar to the *transform feedback* for vertices. Instead of writing the fragment colors into the framebuffer, they are written into a texture. This texture can then be accessed in subsequent render passes. The render to texture methodology is of major importance for deferred shading [Saito and Takahashi, 1990] and also for general purpose computations on GPUs employing shaders.

2.4.2 GPGPU with Shaders

The rendering pipeline of a GPU cannot only be employed for polygonal rendering but also for computing tasks on the rendered results. Examples from computer graphics include post processing effects, like depth of field [Scheuermann and Tatarchuk, 2004; Lee et al., 2009], and deferred shading, e.g. screen-space ambient occlusion [Ritschel et al., 2009]. By adjusting the programmable shaders and enabling the render-to-texture mode, general purpose computations of non-graphics related algorithms are also possible. Examples include particle simulations for SPH [Kolb and Cuntz, 2005] and solving systems of linear equations [Krüger and Westermann, 2003a]. Other typical GPGPU applications are filtering, reduction, and sorting [Pharr, 2005].

When using the rendering pipeline for GPGPU, the shaders represent the computing kernels. Different approaches are possible with the vertex shader, the geometry shader, and the fragment shader. Typically, only the fragment shader is used for general purpose computations. A quad filling the entire viewport is rendered and, hence, the fragment shader is evaluated for each visible pixel resulting in a kernel invocation on a per-fragment level. The necessary input data is stored in textures or given as constants. The texture data is fetched per fragment with the interpolated texture coordinates of the screen-filling quad. Many algorithms such as sorting, filtering, and particle simulations require multiple passes and utilize the *ping-pong rendering* approach. The results of the current pass are stored into a texture with the render-to-texture mode enabled. This texture is then fetched in the subsequent pass as input parameter and the new result is stored into a second texture. Then the first and the second textures are swapped in a ping-pong fashion.

Employing shaders for general purpose computations has several advantages. The number of parallelized kernel invocations is only limited by the maximum available viewport resolution and available texture memory. The kernels make automatically use of texture filtering and caching, making this scheme well suited for algorithms, which naturally fit to the graphics pipeline like image processing. On the other hand, one is restricted to use a graphics API to initiate and perform the calculations. Unstructured or scattered, i.e. random, writing into memory is not possible in the fragment shader where only the current fragment position can be written. This, however, can be circumvented in some case by using the vertex shader. Since the vertex shader only operates on vertices, it can be used to scatter points into the viewport and the computation takes only place for the fragments covered by the footprints of the points. An example is the computation of a density volume from a particle distribution which is described in more detail in Section 5.5.

2.4.3 CUDA

NVIDIA introduced the general-purpose, parallel computing architecture CUDA to harness the parallel computational power of modern graphics processors [NVI, 2011]. With CUDA the developer has the ability to program the GPU with a high-level C-like programming language. The programming model abstracts the hardware to a certain amount and is built around a flexible number of *streaming processors* each equipped with a fixed number of cores, also called *CUDA cores*. The computation of *kernels* is performed in *threads* where one thread represents one instance of the kernel. The threads are coordinated by the CUDA thread hierarchy.

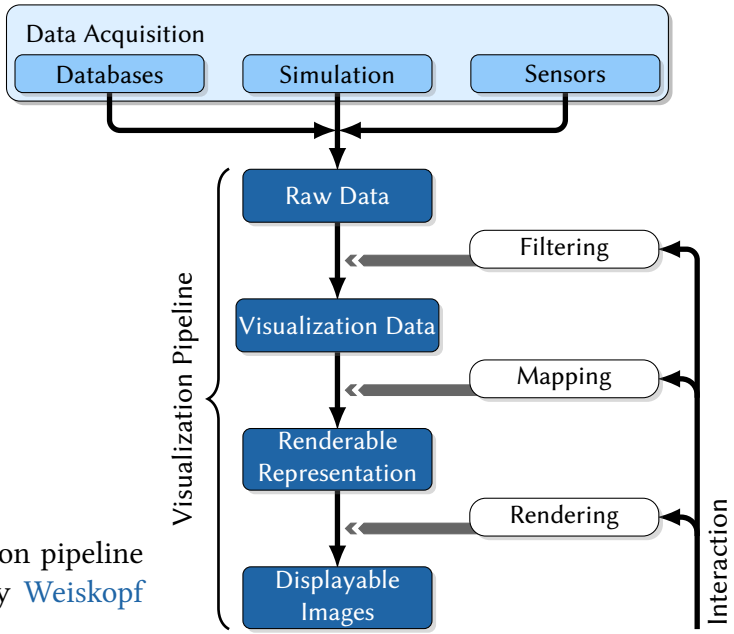
Thread Hierarchy

The compute kernel is executed n times in parallel by n different CUDA threads. The threads can be grouped in one, two, or three-dimensional *thread blocks*. The threads within a thread block are able to cooperate through shared memory and can synchronize their execution. Due to memory limitations the maximum size of a thread block is 1024 on current GPUs. To solve larger problems, the thread blocks have to be arranged in a one- or two-dimensional grid. The blocks in this grid are independent and the CUDA environment distributes them arbitrarily to free multiprocessors for computation. For this reason, the same program can be computed without modifications on every CUDA-capable GPU. A GPU with more multiprocessors will automatically execute the kernel in less time, distributing threads among its multiprocessors.

Memory Layout

In CUDA, the graphics memory can be randomly accessed for both, read and write operations. This allows the usage of a broader class of algorithms and techniques for the compute kernels, e.g. scattering and gathering, in comparison to the usage of GLSL shaders (cf. prev. Section).

A single CUDA thread has access to a small private local memory and can exchange data with other threads of the same thread block over shared memory. Local and shared memory are not persistent and are accessible only by the specific thread or threads of the respective thread block. Global memory, as well as the constant and texture memory, can be accessed by all threads of different kernels. Constant memory contains as the name suggests constants, which cannot be modified by a kernel but only from the CPU side. CUDA textures act as a special, cached memory mapping of array data, in the way that the memory can be accessed as 1D, 2D, or 3D texture with optional filtering similar to texture fetches in the rendering pipeline.



▶ **Figure 2.10** — Visualization pipeline based on the description by Weiskopf [2007].

2.5 Visualization Principles

The concept of a *visualization pipeline* to describe the process of visualizing data was first proposed by Haber and McNabb [1990]. Figure 2.10 depicts their principal idea of the *visualization pipeline* with minor modifications by Weiskopf [2007].

Raw data, the input to the visualization pipeline, is obtained during *data acquisition* from different sources. The data sources might be numerical simulations, databases, or sensors, which are deposited in the environment. The kind of data source implicitly defines the data representation and the time dependency of the data. The *filtering* stage transforms the raw data from the data source into abstract *visualization data* where denoising or smoothing are typical operations. The data might also be enhanced by resampling where missing data values are either interpolated or approximated. Other important operations of the filtering stage are the data segmentation or classification and the selection of data, which might only be a subset or a single slice of a 3D volume. In the *mapping* stage, the abstract visualization data is mapped to attributes of graphical primitives, the *renderable representation*, which extends in time and space. The attributes of the renderable representation can include geometry, color, surface texture, and opacity, which are interpolated in time. Typically, at least predefined colors are assigned to the different segments of the input data using a lookup table. The *rendering stage* generates a single *displayable image* or a sequence of images by performing view transformations and adding shading and scene illumination. It is very rare, that the parameters of the pipeline can be adjusted beforehand to yield an optimal result without any user interaction. Interaction, therefore, plays a crucial role in the visualization pipeline, since it is necessary to explore the large parameter space of simulation and visualization.

In many cases, interactive post processing is used for the visualization process. The data is generated or collected beforehand and stored on permanent storage. Visualization is completely decoupled from the data acquisition and the data is loaded on demand into the visualization pipeline afterward. This scenario is commonly used for exploratory visualization of data, which is unknown, difficult to collect, or the simulation takes a long time to complete. Another approach is *computational steering* [Mulder et al., 1999], also known as *interactive steering* with a tight connection between data acquisition, especially numerical simulations, and the visualization process. The parameters for both, visualization and simulation, can be tweaked interactively during the simulation. Again, interactivity plays an important role concerning response time from the visualization, e.g., when the camera position is changed, or waiting for updated results of the simulation. As a consequence, the need for interactive visualizations and simulations, i.e. efficient algorithms and techniques, arises. The definition of interactive visualization found in literature varies between several seconds per image to several images per second depending on the environment it is used in. When a simulation run takes hours to complete, a user might be satisfied to see intermediate results every one or two minutes. However, if the same user wants to explore the data after the simulation terminated successfully, a movie or an interactive visualization might be preferred.

In this thesis, the specialized architecture of GPUs is harnessed to perform the mapping and the rendering steps for all visualizations and is also used for the stochastic simulation of cellular processes (cf. Chapters 4 and 5). Simulation and visualizations are then combined to a computational steering prototype in Chapter 6. The basic visualization techniques for particle data, which is obtained from stochastic simulation, are described in the following. First, an overview is given on the available commonly used techniques. Two methods, glyph-based rendering and volume rendering, are then explained in more detail as they both provide the foundation of the visualizations in this thesis.

2.5.1 Rendering of Particle Data

There are many particle-based simulation methods from different fields of science, such as MD [McCammon et al., 1977], agent-based simulations [e.g. Klann et al., 2009], discrete element methods, and SPH [Gingold and Monaghan, 1977]. These simulations usually comprise a large number of entities that interact with each other, either in a pairwise interaction scheme or by contributing to a common continuous field which in return affects all entities. The entities can be atoms, molecules, mesoscopic or macroscopic particles and ensembles. From a visualization point of view, all these entities can be handled rather equally as particles.

Particle-based simulations usually generate large amounts of data due to their time-dependent nature. Visualizing such data sets interactively is still a challenge. On commodity workstations, the mere data set sizes require highly optimized software to achieve interactivity. Even the mapping from the individual particles to visual primitives might not be clear. Depending on the focus of the simulation, different visualization methods are commonly used from points or sprite-based particles over density or probability volumes

to highly specialized visualizations like protein secondary structures. While point- or sprite-based rendering directly shows the original data, these images often suffer from high visual complexity due to visual clutter from a high number of discrete graphical primitives. To gain a more compact representation, which is usually easier to understand, the particle data is sampled into a density volume and an isosurface is rendered for a significant isovalue. This approach is related to metaball rendering (also called blobby surfaces) [Blinn, 1982]. The metaball surface is defined by radial symmetric density kernel functions located at each particle. It corresponds to an iso-surface of a density volume. The metaballs can be rendered by ray casting or by extracting a geometric representation, e.g. with Marching Cubes [Lorensen and Cline, 1987]. On recent GPUs, the metaballs can be interactively evaluated and rendered [van Kooten et al., 2007; Müller et al., 2007]. Surface representations are common for protein data sets, in MD, and in quantum mechanics simulations.

2.5.2 Glyph-based Rendering

Glyphs are compact graphical elements visually representing different data attributes and there exist many different glyph shapes like arrows, markings, ellipsoids, etc. For the rendering of glyphs different methods can be employed.

The usage of a polygonal representation for glyphs is straightforward. The shape is approximated with a triangle mesh and then drawn with the rendering pipeline. The resolution of the mesh, however, is important to the smoothness of the rendered result. If the glyph is located close to the viewer and, hence, appears large on the image plane, a high mesh tesselation is required. On the other hand, if the glyph is very far away and small, only a few triangles are necessary. The tesselation shader or the geometry shader might be used to compute the distance-dependent resolution of the mesh and then to refine or coarsen the mesh adaptively. The irregular load of the GPU because of the different glyph tesselations is disadvantageous for this approach. An alternative is instancing with a fixed mesh tesselation, which results in a more steady GPU load. With instancing, a template mesh of the glyph is stored in GPU memory and is instanced and transformed for each actual glyph. The drawback of this method is that a fixed tesselation has to be chosen beforehand unless more elaborate schemes like octrees [Glassner, 1984] or binary space partitioning [Kaplan, 1985] are used for a level-of-detail approach with different instances.

Another approach for glyph rendering is the use of textured billboards. Billboards are screen-aligned quads consisting of only four vertices or a single point if the point sprite extension is used in OpenGL [Bajaj et al., 2004]. The textures for glyphs with different attributes are precomputed and stored in a texture catalog, e.g. for different orientations and lengths of arrow glyphs. Because the texture resolution is fixed, oversampling and undersampling of textures is necessary for glyphs that appear too large or too small in image space, respectively. When using only the texture coordinates of the billboard without a texture fetch, illuminated sphere glyphs can be approximated and, e.g., used to visualize particle data [Green, 2008]. Texture coordinates are used to compute the

surface normal of the sphere and to adjust the depth in the fragment shader. This method is not bound by geometry nor limited by texture resolution since its computations are performed for all fragments in image space. However, the perspective projection of the camera is neglected for the individual glyphs.

Perspectively-correct spherical glyphs can be generated directly on the GPU by solving the implicit sphere equation

$$\|\boldsymbol{x} - \boldsymbol{p}\| - r = 0, \tag{2.3}$$

where $\boldsymbol{p}$ denotes the center of the sphere and r its radius. All positions $\boldsymbol{x}$ which fulfill Equation 2.3 are located on the surface of the sphere.

Gumhold [2003] uses implicitly-described ellipsoids for the visualization of symmetric tensor fields. The surface of the ellipsoids is obtained by transforming them into a local sphere space and performing ray-sphere intersections in the fragment shader. A similar approach was proposed by Klein and Ertl [2004] while reducing the necessary information transferred to the GPU. The general idea of both approaches is to employ the rendering pipeline to perform ray casting. In the fragment shader, a ray R with direction $\boldsymbol{r}_d$ is generated for each fragment starting at the camera position $\boldsymbol{c}$ directed at the fragment's position $\boldsymbol{p}_f$ in object space:

$$R: \quad \boldsymbol{x}(\lambda) = \boldsymbol{c} + \lambda \boldsymbol{r}_d \quad \text{with} \quad \boldsymbol{r}_d = \frac{\boldsymbol{p}_f - \boldsymbol{c}}{\|\boldsymbol{p}_f - \boldsymbol{c}\|} \ \wedge \ \forall \lambda \in \mathbb{R}_{>0}. \tag{2.4}$$

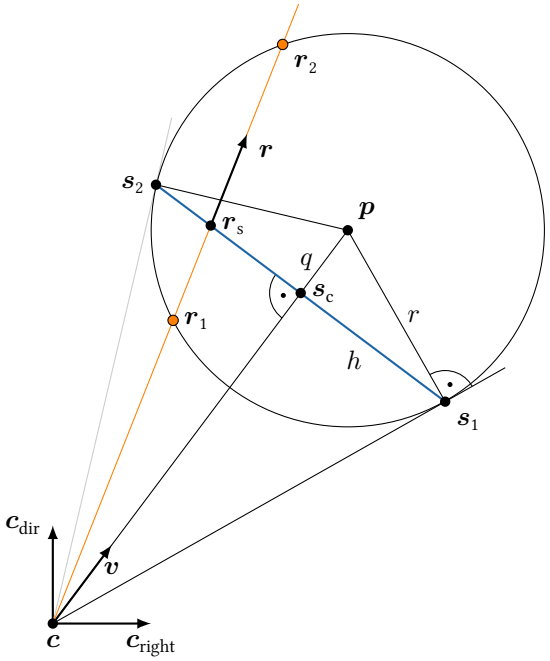
Since only points in front of the camera are of interest, the values of the ray parameter λ have to be positive. Substituting the ray equation 2.4 into (2.3) yields

$$\begin{aligned} & \|\boldsymbol{c} + \lambda \boldsymbol{r}_d - \boldsymbol{p}\| - r = 0 \\ \Leftrightarrow \quad & \left(c_x + \lambda r_{d_x} - p_x\right)^2 + \left(c_y + \lambda r_{d_y} - p_y\right)^2 + \left(c_z + \lambda r_{d_z} - p_z\right)^2 = r^2. \end{aligned} \tag{2.5}$$

Solving the quadratic equation in (2.5) for the parameter λ gives two solutions $\lambda_{1,2}$ for the intersections between the ray and the sphere. Without loss of generality be $\lambda_1 \leq \lambda_2$. Substituting λ in (2.4) then yields the intersection points $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. When rendering opaque spheres, only the smaller λ, i.e. the first intersection point $\boldsymbol{r}_1$, is of interest. The normal $\boldsymbol{n}$ of the sphere at $\boldsymbol{r}_1$ is obtained by

$$\boldsymbol{n} = \frac{\boldsymbol{r}_1 - \boldsymbol{p}}{r}. \tag{2.6}$$

To initiate the ray casting, the fragment shader has to be invoked for the respective fragments covering at least the glyph. This can be done by either drawing a quad filling the viewport for each glyph or drawing a viewport-filling quad once and computing the intersections of all glyphs per fragment. Typically, glyphs are small compared to the viewport size and, hence, both approaches result in a massive overdraw. The drawing complexity can be dramatically reduced when only the footprint of each glyph is drawn with the rendering pipeline. The footprint or silhouette of a glyph is determined by the projection of the visible part of the glyph onto the image plane.



▶ **Figure 2.11** — Silhouette computation of a spherical glyph for the horizontal extent. The blue part visible to the camera at $\boldsymbol{c}$ is bounded by $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ where tangential planes touch the sphere. The intersections between a sample ray $\boldsymbol{r}$ and the sphere are highlighted in orange.

A fast approximation of the silhouette is a tight bounding rectangle, which encloses the glyph in image space. In the vertex shader, the axis-aligned image-space bounding box of the glyph is computed and then projected into image space. The fragment shader is then invoked only within the extents of this bounding box. However, the projection of the bounding box does not necessarily result in a tight shape around the glyph whereas approximating the glyph's silhouette gives better results.

Without loss of generality, the following silhouette approximation uses four points on the spherical surface to confine the silhouette. The silhouette of spherical glyphs is always elliptical and lines connecting the silhouette with the camera at $\boldsymbol{c}$ are tangential to the sphere. A four-sided pyramid with apex in $\boldsymbol{c}$ is composed of four planes touching the silhouette of the sphere at the horizontal and vertical extreme values yielding a tight bounding box (see Figure 2.11).

The problem of determining the four tangential points in 3D can be decomposed into two 2D problems, one horizontal and one vertical. Figure 2.11 shows the geometric solution for the horizontal axis. First, the view direction $\boldsymbol{v}$ from the camera at $\boldsymbol{c}$ towards the glyph center $\boldsymbol{p}$ is determined by

$$\boldsymbol{v} = \frac{\boldsymbol{p} - \boldsymbol{c}}{\|\boldsymbol{p} - \boldsymbol{c}\|}. \tag{2.7}$$

Let $\boldsymbol{s}_{1,2}$ be the tangential points limiting the horizontal extent of the silhouette. By applying the Euclidean theorem of sides to the right triangle $\triangle \boldsymbol{c}\boldsymbol{s}_1\boldsymbol{p}$ gives the distance $q = \|\boldsymbol{s}_c - \boldsymbol{p}\|$ between $\boldsymbol{p}$ and the silhouette's center $\boldsymbol{s}_c$

$$q = \|\boldsymbol{s}_c - \boldsymbol{p}\| = \frac{r^2}{\|\boldsymbol{p} - \boldsymbol{c}\|}. \tag{2.8}$$

Using the Euclidean theorem of height, the distance $h = \|\boldsymbol{s}_1\boldsymbol{s}_c\|$ is given by

$$h = \sqrt{q(\|\boldsymbol{p} - \boldsymbol{c}\| - q)}. \tag{2.9}$$

With (2.8) and (2.9) given, the tangential points $\boldsymbol{s}_{1,2}$ are determined by

$$\boldsymbol{s}_{1,2} = \boldsymbol{p} - q\boldsymbol{v} \pm h\boldsymbol{v}', \tag{2.10}$$

where $\boldsymbol{v}'$ is orthogonal to $\boldsymbol{v}$. The tangential points of the vertical case are determined in a similar way. The size of footprint is then determined from all four points and results in a tighter shape than the projection of the bounding box.

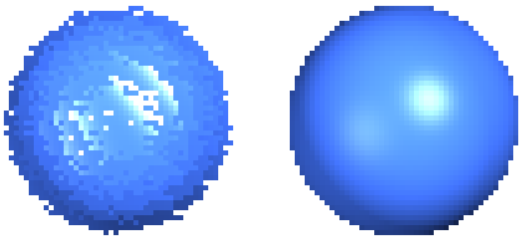
For the rendering of a single glyph, only a single point has to be drawn. The vertex shader calculates the extent of the footprint and adjusts the point size accordingly. The fragment shader then performs the ray casting step. The necessary information, which has to be transferred to the GPU, consists of the glyph's position $\boldsymbol{p}$, its radius r, and additional attributes like color instead of a triangle mesh consisting of several hundreds of vertices. Instead of using point primitives, the geometry shader can be used to generate a tighter silhouette geometry as input for the ray casting instead of quadrilateral point splats [Klein, 2008; Grottel et al., 2009].

So far, only four tangential planes are used to approximate the silhouette of spherical glyphs. Klein [2008] tested different numbers for ellipsoidal glyphs. In his tests, a hexagon was the optimal shape for the trade-off between computational costs and saved fragment computations. The concept of ray casted glyphs is also applicable to other quadratic primitives, like cylinders and cones. Based on the same method, Reina and Ertl [2005] showed how to render dipoles constructed from spheres and cylinders. This approach was extended by Grottel et al. [2009] to arbitrarily composed glyphs.

When computing the ray direction and calculating the intersection in the fragment shader, the glyphs may start to look fuzzy due to floating point precision issues on the GPU. This effect is emphasized for small glyphs in large scenes or camera settings with a small aperture angle. Figure 2.12 shows a glyph with those type of artifacts on the left and the correct result on the right. The artifacts are mainly caused by the small differences of the un-normalized ray direction $\boldsymbol{r}$ between neighboring fragments. To prevent those artifacts, it is sufficient to move the ray origin towards the glyph. As the position $\boldsymbol{r}_s$ on the glyphs silhouette (cf. Figure 2.11) is given by the inverse transformation of the fragment position into object coordinates it can be used as shifted ray origin. Adjusting to the shifted ray origin (cf. (2.4)) results in

$$R_{\text{shifted}}: \quad \boldsymbol{x}(\lambda) = \boldsymbol{r}_s + \lambda\boldsymbol{r}_d \quad \forall\lambda \in \mathbb{R}. \tag{2.11}$$

Since the first intersection point $\boldsymbol{r}_1$ now lies behind the ray origin, λ_1 is negative. Care has to be taken when the first intersection point lies behind the camera, i.e. $|\lambda_1| > \|\boldsymbol{r}_s - \boldsymbol{c}\|$. In this case, the glyph has to be either discarded or the interior is to be revealed and the depth has to be adjusted accordingly.



▶ **Figure 2.12 —** Artifacts due to floating point precision on the GPU can occur if the ray origin lies in the camera $\boldsymbol{c}$ and the glyph is small (left). When the ray origin is moved into the glyph the artifacts disappear (right).

2.5.3 Volume Rendering

A common technique for visualizing 3D scalar fields is volume rendering. Volume rendering is mainly applied but not limited to the field of medical imaging creating images of magnetic resonance tomography (MRT) or computed tomography (CT). Other examples can be found in different areas of engineering and science, including computational fluid dynamics, or geology.

In the stationary case, a 3D scalar field is a map s which maps a position $\boldsymbol{x}$ onto a scalar value

$$s : \Omega \longrightarrow \mathbb{R}, \quad \boldsymbol{x} \longmapsto s(\boldsymbol{x}), \tag{2.12}$$

where $\Omega \subset \mathbb{R}^3$ denotes the 3D Euclidean space.

A two-dimensional image of the scalar field is synthesized by modeling the transport of light along a viewing ray. Under the assumption that the scalar field is a gaseous volume, i.e. a participating medium, light interacts with the medium in several ways. Particles in the medium can absorb the incoming light or emit light, and the light might be scattered. Hence, the contribution to the radiation at a single point is expressed as the sum of several components that include absorption, emission, and in- and out-scattering.

By considering only emission and absorption, the equation of light transfer can be simplified to the emission-absorption model or density-emitter model [Sabella, 1988]. Given a ray R along the viewing direction $\boldsymbol{v}_{\text{dir}}$

$$R: \quad \boldsymbol{r}(\lambda) = \boldsymbol{r}_{\text{start}} + \lambda \boldsymbol{v}_{\text{dir}} \quad \text{with} \quad \forall \lambda \in \mathbb{R}, \tag{2.13}$$

the change of the radiance L for the emission-absorption model at position λ along ray R can be formulated as

$$\mathrm{d}L(\lambda) = g(\lambda) - \kappa(\lambda) L(\lambda)\, \mathrm{d}\lambda. \tag{2.14}$$

The amount of emitted light at position λ is given by the source term $g(\lambda)$. The extinction coefficient $\kappa(\lambda)$ defines how much of the incoming light is attenuated by the participating medium. The source term and extinction coefficient, are determined for each sample point in the 3D scalar field by two transfer functions $\tilde{\kappa}$ and $\tilde{c}$. Under the assumption of a local thermodynamic equilibrium, the emission coefficient is equal to the product of

the absorption coefficient and the spectral radiance according to Planck's law. Thus, the extinction coefficient $\kappa(\lambda)$ and the source term $g(\lambda)$ are given by

$$\kappa(\lambda) = \tilde{\kappa}\big(s(\boldsymbol{r}(\lambda))\big) \quad \text{and} \tag{2.15}$$

$$g(\lambda) = \tilde{\kappa}\big(s(\boldsymbol{r}(\lambda))\big)\tilde{c}\big(s(\boldsymbol{r}(\lambda))\big), \tag{2.16}$$

where the transfer function $\tilde{c}$ is scaled accordingly. Integrating Equation 2.14 for a viewing ray starting at the initial point at $\lambda = \lambda_0$ and ending at $\lambda = \xi$ yields the *volume rendering integral* [Max, 1995]

$$L(\xi) = L(0)\mathrm{e}^{-\int_{\lambda_0}^{\xi} \kappa(\lambda')\,\mathrm{d}\lambda'} + \int_{\lambda_0}^{\xi} g(\lambda)\mathrm{e}^{-\int_{\lambda}^{\xi} \kappa(\lambda')\,\mathrm{d}\lambda'}\,\mathrm{d}\lambda. \tag{2.17}$$

With $L(0)$ as radiance at the ray start, i.e. $\lambda = \lambda_0$, the accumulated radiance leaving the volume is then given by $L(\xi)$. The definition of the optical transparency,

$$T(\lambda_1, \lambda_2) = \mathrm{e}^{-\int_{\lambda_1}^{\lambda_2} \kappa(\lambda')\,\mathrm{d}\lambda'}, \tag{2.18}$$

allows to rewrite the volume rendering integral into the shorter form

$$L(\xi) = L(0)T(\lambda_0, \xi) + \int_{\lambda_0}^{\xi} g(\lambda)T(\lambda, \xi)\,\mathrm{d}\lambda. \tag{2.19}$$

The volume integral in (2.19) is usually approximated numerically by a Riemann sum over n equidistant segments. The length of each segment is then given as $\Delta\lambda = (\xi - \lambda_0)/n$. The approximation is carried out in several steps. The optical transparency (Equation 2.18) between segment i at $\lambda_i = \lambda_0 + i\Delta\lambda$ and the end point at $\lambda = \xi$ approximated by a Riemann sum yielding

$$T(\lambda_i, \xi) \approx \mathrm{e}^{-\sum_{k=i}^{n-1} \kappa(\lambda_k)\Delta\lambda} = \prod_{k=i}^{n-1} \mathrm{e}^{-\kappa(\lambda_k)\Delta\lambda}. \tag{2.20}$$

Hence, the transparency of the ith segment is given by

$$t_i = \mathrm{e}^{-\kappa(\lambda_i)\Delta\lambda}. \tag{2.21}$$

By defining the source term g_i for the ith segment as

$$g_i = g(\lambda_i)\Delta\lambda \tag{2.22}$$

and the definition of the transparency in (2.21), the volume rendering integral in (2.19) can be approximated for n segments with

$$L(\xi) \approx L(0)\prod_{i=0}^{n-1} t_i + \sum_{i=0}^{n-1} g_i \prod_{j=i+1}^{n-1} t_j. \tag{2.23}$$

The discretized integral is used in different variations for direct volume rendering (DVR). The rendering methods can be classified as object-order or image-order approaches.

Object-order techniques sample the volumetric data in the object space of the underlying data grid. The contribution of each single grid cell is computed and then projected onto the image plane. Texture-based slicing as proposed by Westermann and Ertl [1998] is an early GPU implementation of volume rendering. Here, object-space-aligned texture stacks are blended directly, harnessing the capabilities of GPUs. Other approaches for object-order direct volume rendering include splatting [Westover, 1990], the shear-warp factorization [Lacroute and Levoy, 1994], and projected tetrahedra [Shirley and Tuchman, 1990]. Image-order techniques in contrast perform a sampling of the 2D image plane. Instead of computing the contribution of each grid cell, the radiance leaving the volume is determined for each pixel on the image plane by ray casting [Drebin et al., 1988; Levoy, 1988; Sabella, 1988]. When ray casting is employed Equation 2.23 can directly be evaluated per pixel. In the following, the most prominent volume rendering techniques, texture-based slicing and ray casting, are described in more detail since they are used for the visualization methods presented in Chapter 5 (Sections 5.5 and 5.6).

Texture-based Slicing

Slicing belongs to the texture-based visualization methods [Cabral et al., 1994; Xue et al., 2005].Since current graphics hardware provides no support for volume primitives (cf. Section 2.4.1), a proxy geometry is needed. The proxy geometry consists of a set of rendering primitives that represent the volume. Usually, the volume is cut into several slices, each slice is textured with data of the volume data set, and all slices are rendered.

If the proxy geometry is rendered from back to front also known as over-operator [Porter and Duff, 1984], can be employed. The color C'_i of the ith slice, $i \in \{0, \ldots, n-1\}$, is then computed by back-to-front-compositing:

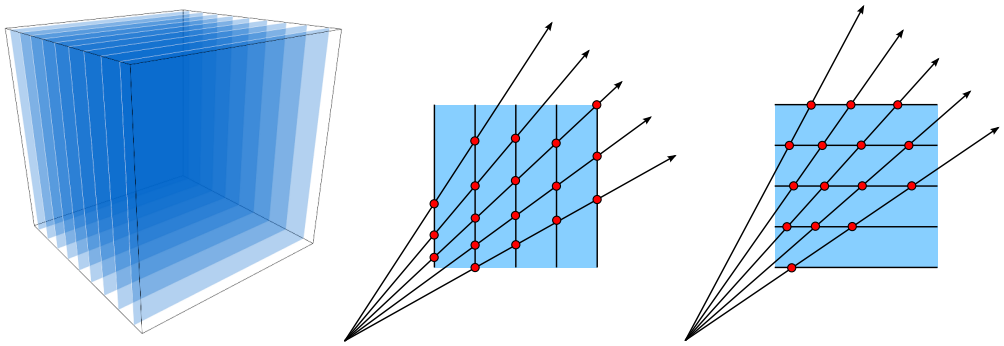
$$C'_i = (1 - \alpha_i)\, C'_{i+1} + \alpha_i C_i, \tag{2.24}$$

where $C'_n = 0$ and C'_i and α_i correspond to the color and opacity of slice i, respectively. When the rendering order is reversed, the compositing order has also to be reversed. Front-to-back compositing requires that the computed opacity value is stored explicitly in the framebuffer besides the color. With the initial values for color and opacity set to $C'_0 = 0$ and $\alpha'_0 = 0$, the compositing can be expressed as follows

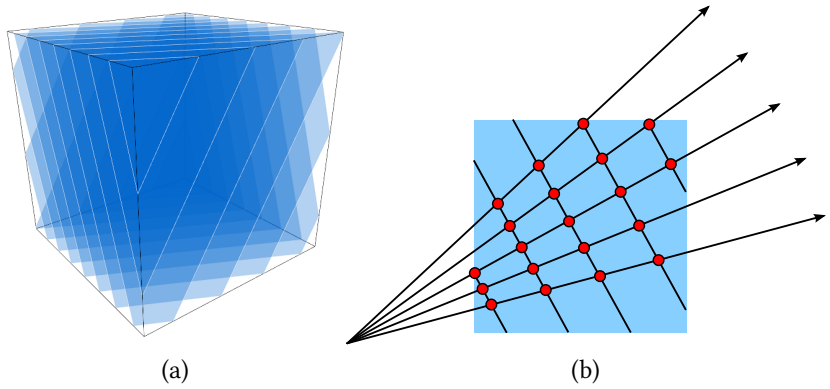
$$C'_i = C'_{i-1} + (1 - \alpha'_{i-1})\, \alpha_i C_i \quad \text{and} \tag{2.25}$$

$$\alpha'_i = \alpha'_{i-1} + (1 - \alpha'_{i-1})\, \alpha_i. \tag{2.26}$$

For the generation of the proxy geometry, i.e. the volume slices, two principles are available. The first one aligns the cut planes along the main axes of the volume and is, therefore, called *object-aligned slicing*. For each principal axis a stack of 2D textures is generated from the volume, which is then used for texturing the proxy geometry of each cut plane (see Figure 2.13(a)). The principal axis with the lowest deviation from the viewing direction is chosen and the attached proxy geometry is rendered. Drawbacks of this method are that only bilinear interpolation can be used on each texture slice and



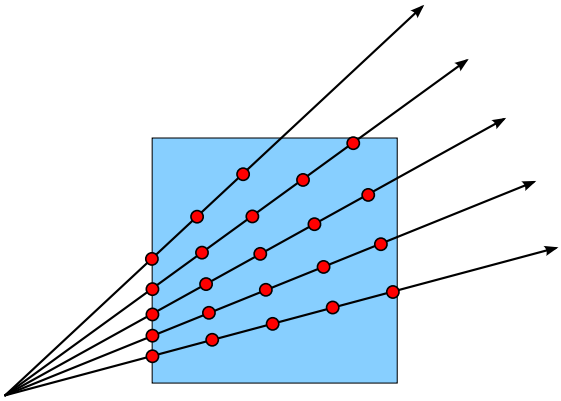
▲ **Figure 2.13 —** Axis-aligned slicing for direct volume rendering. (a) Layout of the slices inside the bounding box of the volume. (b), (c) Sampling positions (red) and reordering of slices for two distinct camera positions.



▲ **Figure 2.14 —** View-aligned slicing for direct volume rendering. (a) Layout of the slices inside the bounding box of the volume. The slices are oriented towards the camera. (b) Location of the sampling positions (red) on the slices.

three times the memory is required because of the three texture stacks. In addition, the distance between sampling positions is not uniform when the camera position is changed. Figure 2.13(b) and (c) illustrates the movement of sampling positions when the viewing axis is changed and the texture stack is exchanged. The perceived brightness might change significantly because the positions of the samples are different.

To overcome the brightness issues, the proxy geometry can be oriented along the viewing direction (*view-aligned slicing*) requiring the volume to be stored in a 3D texture. The texture coordinates of each proxy plane are obtained by cuts with the bounding box of the volume. The view-aligned texture stacks provide higher visual quality due to trilinear interpolation inside the volumetric data. The sampling density can easily be adjusted by increasing or decreasing the number of slices. An example layout of the slices inside the bounding box and the sample positions on the slices are shown in Figure 2.14. The visual quality can further be improved by employing pre-integration [Engel et al., 2001].



▶ **Figure 2.15** — Ray casting for direct volume rendering (DVR). Rays cast into the volume are sampled equidistantly (indicated in red) to solve the volume rendering integral.

Ray Casting

In contrast to texture-based slicing, ray casting belongs to the image-order techniques of DVR. In 1988, volume ray casting was presented independently by Drebin et al. [1988], Levoy [1988], and Sabella [1988] and bears resemblance to ray tracing. The image obtained from both ray casting and ray tracing is computed by casting rays from the virtual camera through every pixel of the viewport. In contrast to traditional ray tracing, secondary rays, i.e. reflection, refraction, and shadow rays, are not considered in volume ray casting. Additionally, the volume has to be sampled along each ray since there is no explicit geometry given to perform ray-geometry intersections inside the volume. The equidistant sampling along the viewing rays allows to directly evaluate the discretized volume rendering integral given in (2.23). Figure 2.15 depicts the sampling positions for several rays inside the volume. At each sampling position a lookup into the volumetric data is performed and a transfer function is used to compute the source term and the extinction coefficient from the interpolated scalar values (cf. Equation 2.15).

Since the computation of each ray is completely independent, this method is well-suited for the SIMT architecture of GPUs. Krüger and Westermann [2003b] presented a view-aligned ray casting approach employing programmable graphics hardware. These multi-pass rendering approaches have been superseded by single-pass volume ray casting, presented by Hadwiger et al. [2005] and Stegmaier et al. [2005]. Here, the ray casting is performed in a single fragment shader. To initiate the rendering, the bounding box of the volume is rendered as proxy geometry. The object space coordinates of the bounding box vertices are interpolated for each render primitive and are then used in each covered fragment to set up a ray. The individual rays are sampled equidistantly in the fragment shader until they leave the bounding box. The volumetric data, stored in a 3D texture, is accessed at the current sampling position, trilinearly interpolated, and the transfer functions are applied. The intermediate result is combined with previously obtained results by front-to-back compositing.

In addition to the volumetric visualization obtained by ray casting, isosurfaces can be determined and rendered by slightly adapting the sampling loop in the fragment shader. If two subsequent samples along a ray lie on different sides of the isosurface, the intersection between ray and isosurface can be obtained by linear interpolation. The isosurface is

shaded and blended with the volume rendering. Spatial perception is improved by means of illumination, e.g. applying the model of Phong [1975]. The normal needed for the illumination computation corresponds to the gradient of the volume at the sampling position and is either obtained in the shader, e.g., by applying central differences or performing a lookup in a precomputed gradient texture. An overview of volumetric ray casting combined with illumination effects is given by Hadwiger et al. [2008].

The volume rendering with ray casting can be accelerated by different means. One simple solution is to terminate the calculation of a single ray when the accumulated opacity exceeds a certain threshold. This method is known as *early ray termination* [Levoy, 1990; Danskin and Hanrahan, 1992]. Another technique is *empty space skipping* [Li et al., 2003] where the sampling positions between the start position of a ray and the first contributing sample are omitted. Usually, the acceleration structure however needs to be rebuilt whenever the transfer function changes and, hence, the first contributing sample might change. Depending on the structure of the volumetric data and the chosen transfer function these two methods provide a significant speedup with almost no additional overhead.

CHAPTER 3

VIRTUAL CELL MODEL

The virtual cell model is a three-dimensional, simplified, mesoscopic model of a cell. The model consists of the plasma membrane defining the boundary of the simulation domain as well as the nucleus, other compartments, and molecules. The complex interior of the cell is reduced to elements actively or passively participating in the signal transduction processes to be studied. To simulate the crowded cellular environment, filaments of the cytoskeleton and static molecules, e.g., ribosomes, are included. The cytoskeletal filaments comprise both microtubules and microfilaments that can be used for simulating the motorized transport mechanisms. Furthermore, nuclear pore complexes are modeled to allow for the import and export of molecules into the nucleus.[1]

Regarding the time scale on which the signal propagation takes place, i.e. seconds to minutes, the structural changes of the cell are only marginal. The modeled structures of the cell are, therefore, static with the exception of the microfilaments and the dynamic proteins used for modeling the signaling and transport processes. The proteins are handled as individual agents and are self-contained, i.e. proteins are considered as one unit instead of thousands of atoms. A novel particle-based simulation is used to simulate the massive number of agents occurring in a cell. In the remaining parts of this thesis, agents and particles are treated to be synonyms. The particles themselves move through the cellular environment and can interact with the environment or each other in biochemical reactions. The particle simulation itself uses an Euler-Maruyama integration scheme to solve the SDEs numerically.

The microfilaments of the cytoskeleton are important to maintain and adjust the shape of the cell. By directed elongation of the microfilaments the cell is able to move, e.g.,

[1] **Parts of this chapter have been published in**: M. Falk, M. Ott, T. Ertl, M. Klann, and H. Koeppl. Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology. In *International Conference on Computational Methods in Systems Biology (CMSB 2011)*, pages 73–82, 2011.

toward a food source through the process of chemotaxis. The growth and shrinking of the filaments is the result of a polymerization process of the subunits of the filaments [Guo et al., 2009; Gruenert et al., 2010]. For the chosen scale simple elongating is sufficient. The growth of the filaments can be simulated by adjusting their actual length depending on local protein concentrations. The movement of proteins themselves and the assembly of the cytoskeleton can trigger local differences in the protein concentration. Another source for those differences are external stimuli like for example a nutrition gradient in the extracellular environment. The cell responds to those stimuli through membrane-bound proteins that arrange themselves according to the gradient, which in turn affects the growth of the microfilaments inside the cell.

The underlying theoretical model for the simulation is presented in the subsequent sections together with a graphical notation representing the virtual cellular model. Aspects of the parallel implementation can be found in Chapter 4.

3.1 Mechanistic Model

The original spatial agent-based cell model was developed by Lapin et al. [2007] and was further improved by Klann [2011]. The model incorporates a model cell environment including the cytoskeleton, various chemical reactions between proteins of first and second order, and transport mechanisms by diffusion and motor proteins. The cytoskeleton is constructed of cylinders with the length and diameter of the corresponding cytoskeletal filaments, i.e. microfilaments and microtubules, which occupy the desired volume fraction of the cell. The model is evaluated with a three-dimensional, discrete MC simulation. The particle-based simulation includes all molecules of a given (bio-)chemical reaction network and tracks their positions in the cell. By modeling the cellular interior explicitly and taking it into account during the simulation, fluctuations in the protein distributions become visible. Thus, the virtual cell model resembles the actual cell and results of simulations resemble more closely to wet lab experiments than, e.g., simulations without spatial structures.

The basic model is extended by adding nuclear import and export through the nuclear pore complexes and emulating the response to external stimuli by adjusting the microfilaments dynamically during the simulation. Individual components of the model are described in more detail in the following sections.

3.1.1 Particles

Every dynamic molecule or protein of interest that is participating in the simulation is represented by a single spherical particle embedded in a fluid. The radius of every particle is set to the corresponding hydrodynamic radius of the original molecule enclosing it completely. The hydrodynamic radius can be obtained from the diffusion coefficient and the viscosity of the fluid with the Stokes-Einstein equation [Einstein, 1905]. In addition to

the radius, the particle also holds its position $\boldsymbol{x}$ inside the three-dimensional simulation domain, its molecular species, and a unique identifier. The identifier is mainly used in the analysis step following the simulation. During analysis and visualization, the identifier allows the tracking of individual particles and to display their trajectories. The molecular species is important for both, simulation and analysis.

The species of the molecule determines several key properties of the particle and can also be used for data segmentation in the subsequent visualizations. Besides the radius, the molecular species implicitly defines the diffusion coefficient D_i for molecule movement and the reactions the particle can take part in.

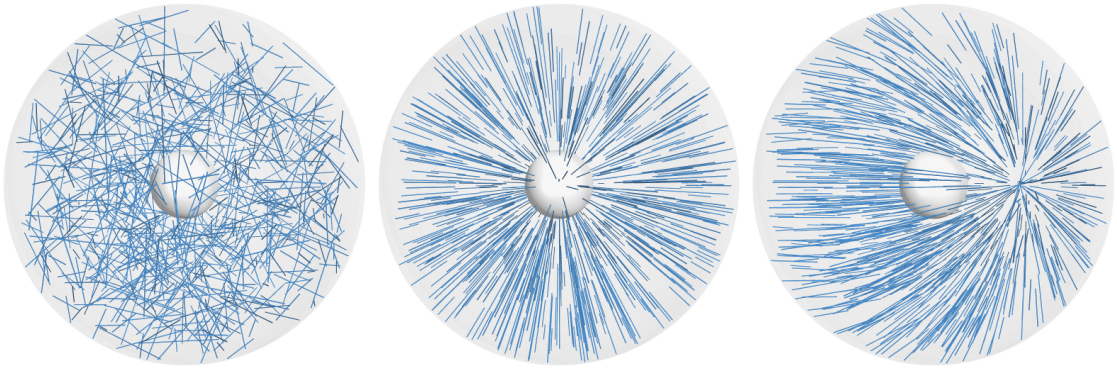
The molecular species, their respective properties, and the abundance and initial distribution in the cell have to be provided by the user as well as the reactions between them and the properties of the model cell itself. During the setup of the simulation, the particles are randomly distributed in the simulation domain, i.e. inside the spherical plasma membrane, according to their initial location. Particles can be initially defined as membrane-bound or diffusing. To account for the fact that some molecules are not found in certain areas of the cytoplasm, particle positions can be constrained. The positions of a specific species can be limited to be inside a certain spherical shell that is given by a minimal and a maximal radius. This allows, e.g., the modeling of the MAPK cascade, where MAP3K is located near the plasma membrane due to scaffold proteins (cf. Section 2.1.2 and [Kholodenko, 2003]).

3.1.2 Cellular Structures

Structures like the plasma membrane, the nuclear envelope, and the cytoskeletal filaments are handled as geometric objects, which can be described implicitly by quadratic equations. The plasma membrane and the nuclear envelope are both spherical with predefined radii. The spherical symmetric shape of the cell was chosen because it is easier to set up than asymmetric cells and it simplifies the comparison between different simulations. The elements of the different types of cytoskeleton are approximated by elongated cylinders.

The nucleus can be arbitrarily positioned inside the cell as long as it is completely enclosed by the plasma membrane. Without loss of generality, the displacement of the nucleus is restricted onto one major axis because of the symmetric hull of the plasma membrane. The diameters of plasma membrane and nucleus, or to be more specific the nuclear envelope, are both user-defined. The radius of the plasma membrane is given by r_{mem}, the radius of the nuclear envelope by r_{nucl}.

For the elements of the cytoskeleton different placement methods are available. The first and most simple method is the random positioning of filaments. The second placement method arranges the filaments around the nucleus in a radial fashion. The MTOC, or centrosome, is implicitly modeled and assumed to be near or inside the nucleus, which is biologically not valid but allows one to model symmetric cells. The filaments can optionally be anchored at the plasma membrane or the nuclear envelope, i.e. one end remains fixed and the other one extends into the cytoplasm. With the polarized placement, the filaments are focusing on a predefined point, e.g. the MTOC. An electric field surrounding



(a) (b) (c)

▲ **Figure 3.1** — Placement of 800 cytoskeletal filaments. (a) random placement. (b) radial toward the nucleus. (c) polarized following field lines modeled by artificial electric dipole (the second point charge is located on the left side of the plasma membrane).

two point charges is used to lay out the filaments along the field lines. If the two point charges are positioned properly, the field lines match the layout of the microtubules of a eukaryotic cell bending around the nucleus and coming together at the MTOC.

All three placement methods are depicted in Figure 3.1. Both radial and polarized placement can be employed for the filament positioning reflecting the cellular structures of eukaryotic cells constituted by microtubules. The random placement can be used for modeling microfilaments but could also be used to study, for example, the effects on the motorized transport mechanism. The polarity of the single filaments is inherently encoded by the direction of the filament. The filament end directed toward the nucleus is the negative one, so that it corresponds to the traveling direction of dynein (cf. Section 2.1.1). The lengths of the filaments are either fixed length or variable length with a given minimal and maximal length. The last remaining parameter that is necessary to define the cytoskeletal filaments, is the radius of the cylinder. This radius is set once per filament type and used for all elements of this class.

In addition to the aforementioned structures, static molecules can be placed randomly in the cellular model. The molecules are modeled with spheres like the particles representing the dynamic molecules. The inclusion of static molecules is necessary to simulate the effects of molecular crowding and hindered diffusion because in our proposed model particle collisions between dynamically diffusing molecules are not inherently included.

3.1.3 Diffusion in the Cytoplasm

The molecules inside a cell are suspended in the cytoplasm. The movement of the molecules originates from collisions with other molecules of the solvent resulting in a Brownian motion. A stochastic simulation supports diffusion of the dynamic molecules by a random walk step mimicking Brownian motion. In every time step of the simulation, a molecule moves according to a random walk, where the step length depends on its

diffusion coefficient D_i and the time step Δt [Klann et al., 2009]. Given the position $\boldsymbol{x}_i$ of molecule i at time t, the updated position $\boldsymbol{x}_i(t + \Delta t)$ is given by

$$\begin{aligned} \boldsymbol{x}_i(t + \Delta t) &= \boldsymbol{x}_i(t) + \Delta \boldsymbol{x}_i \quad \text{with} \\ \Delta \boldsymbol{x}_i &= \sqrt{2 D_i \Delta t}\, \boldsymbol{\xi}_{\mathcal{N}}, \end{aligned} \tag{3.1}$$

where $\boldsymbol{\xi}_{\mathcal{N}} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, I)$ is a normal-distributed 3D random vector with zero mean and a 3×3 identity matrix as co-variance matrix. Applying the central limit theorem allows to approximate the normal distribution with a uniform distribution with $\boldsymbol{\xi}_{\mathcal{U}} \sim \boldsymbol{\mathcal{U}}(-\sqrt{3} \cdot \mathbf{1}, \sqrt{3} \cdot \mathbf{1})$ and $\boldsymbol{\xi}_{\mathcal{U}}$ being a uniformly distributed random vector between $-\sqrt{3} \cdot \mathbf{1}$ and $\sqrt{3} \cdot \mathbf{1}$ [see Klann, 2011, Chap. A.1].

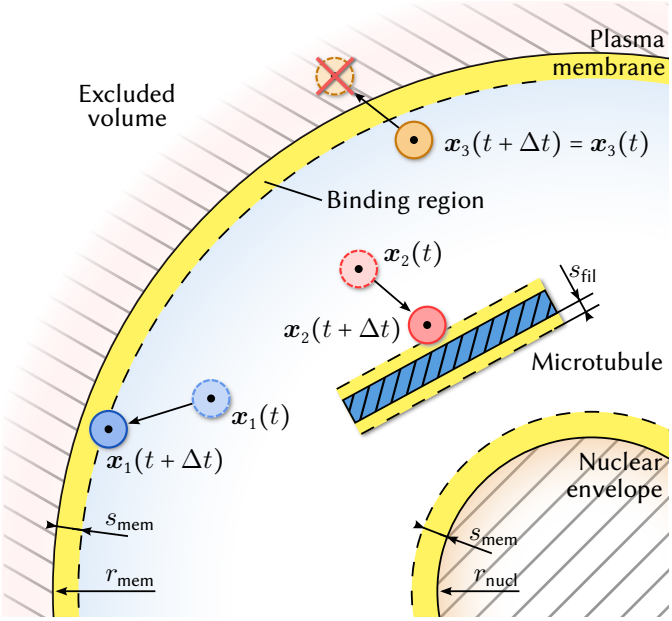
The movement of molecules bound to a membrane is restricted to the respective surface, i.e. the plasma membrane or the nuclear envelope, and, hence, two-dimensional. Instead of the free diffusion according to (3.1), the diffusion is restricted to a disk perpendicular to the surface normal at the current position. A 2D random vector is sampled and the new position $\boldsymbol{x}_i(t + \Delta t)$ is projected back onto the surface afterward.

The rotational diffusion of the individual molecules is not included because of the rotational symmetry of the spherical representations. In comparison to the translational diffusion, the rotational diffusion is much faster [Partikian et al., 1998]. Considering this, the particles can be assumed to represent the average of different rotations at a certain time step t. When modeling the atomistic scale, the rotational diffusion has to be considered for correct bounds between proteins.

The diffusion process of the particles is hindered by the structural elements residing in the cytoplasm, i.e. the cytoskeleton and static molecules, effectively reducing the diffusion. Different approaches exist to handle particle collisions with objects. The first approach is physically motivated and the calculation of the collision is based on impacts. Of all possible collision objects, the closest one is determined by regarding the direction of the particle movement $\Delta \boldsymbol{x}$ (Equation 3.1). The impact time and the resulting reflected position of the particle are computed. The trajectory between impact position and the reflected one, however, has to be checked for other possible collisions. In a crowded environment and a large step width, this approach demands a high computational effort and might lead to load-imbalances among the particles which could hamper a parallel simulation.

Another approach lies in the choice of a new particle position in the case of a collision. The new position us tested again for collisions with all other cellular structures. If the particle at the updated position collides with any object, a new position with a different step $\Delta \boldsymbol{x}$ has to be determined (cf. Equation 3.1). If the number of trials is not limited, the approach could result in an infinite amount of computation in crowded environments.

The third technique is to simply discard the invalid position. If the new position of a particle lies outside the simulation domain or inside a cellular structure, this new position will be rejected and the old position is reused. The computational overhead is reduced to a minimum, i.e. the collision tests have to be performed only once, and it could be argued that the result is still a Brownian motion [Trinh et al., 2000; Klann, 2011]. This



◀ **Figure 3.2 —** Interactions between cellular structures and molecules. Molecules can bind to cellular structures if they intersect with the binding region (yellow). The hatched areas indicate regions where molecules are prohibited, i.e. where updated positions are immediately rejected.

approach is most suitable for a parallel simulation and is used for the model of the virtual cell presented in this thesis. Interactions and the handling of collisions between diffusing particles and structures are described in the next section.

3.1.4 Interactions with Cellular Structures

In the context of this work, interactions with structures are considered to be collisions of particles with the static cellular structures and the attachment of particles onto structures. The difference between a collision and a potential attachment is that a collision occurs if the spherical particle intersects with the geometry of the cellular structure. The attachment of a particle, on the other hand, is possible when the particle is located inside the close vicinity of the structural element. If the particle intersects with a reaction volume surrounding a structure, adsorption is possible. The reaction volume is defined by the surface of the structure and the binding distance s_s extending perpendicularly to the surface s. Assuming mass action kinetics [Waage and Guldberg, 1864], interactions between the reaction volume of surface s and molecules of type i are dependent on a rate constant. The rate constant k_{si} ([µm/s]) and the time step Δt can be used in turn to express the binding distance

$$s_s = k_{si} \Delta t. \tag{3.2}$$

The smaller the value of Δt, the smaller is the binding distance s_s yielding a constant binding rate independent of the chosen time step. The binding distance of the elements of the cytoskeleton is defined by s_{fil}. The distances for the plasma membrane and the nuclear envelope are identical and are given by s_{mem}. Figure 3.2 illustrates the different binding regions of microtubules and membranes. The hatched areas indicate where molecular positions are rejected immediately [Klann, 2011].

Given the definitions of s_{mem} and s_{fil} by (3.2), the necessary conditions for a possible interaction can be defined as follows. Attachment of a particle on the plasma membrane is possible, if the minimal distance between particle and plasma membrane is smaller than s_{mem}, i.e. the following inequality is fulfilled

$$r_{\text{mem}} - \left\| \boldsymbol{x}_i - \boldsymbol{x}_{\text{mem}} \right\| \leq r_i + s_{\text{mem}}, \tag{3.3}$$

where $\boldsymbol{x}_i$ is the validated particle position, i.e. not inside the collision volume, r_i is the particle radius, and $\boldsymbol{x}_{\text{mem}}$ is the position of the spherical plasma membrane. For the binding region outside the nuclear envelope located at $\boldsymbol{x}_{\text{nucl}}$, the signs on the left have to be exchanged yielding

$$-r_{\text{nucl}} + \left\| \boldsymbol{x}_i - \boldsymbol{x}_{\text{nucl}} \right\| \leq r_i + s_{\text{mem}}. \tag{3.4}$$

Based on (3.3), the interaction between a particle i and a filament f can be defined as

$$d \leq r_i + r_f + s_{\text{fil}}, \tag{3.5}$$

with r_f as the radius of the filament f and d being the minimal distance between f and the particle located at $\boldsymbol{x}_i$. Without the loss of generality, the shortest distance d between a point $\boldsymbol{x}_i$ and an infinite line is given by

$$d = \frac{\boldsymbol{d}_f \times (\boldsymbol{x}_f - \boldsymbol{x}_i)}{\left\| \boldsymbol{d}_f \right\|}, \tag{3.6}$$

where $\boldsymbol{x}_f$ is the center point of the filament and $\boldsymbol{d}_f$ its direction and $\times$ represents the vector cross product. Care has to be taken for a filament with finite length, so that the projection of $\boldsymbol{x}_i$ onto the filament lies inside the cylinder.

If a particle is close to a structure s but is still located inside the accessible regions, an interaction between both is possible when defined in the underlying model. Assuming a uniform distribution of interaction events in the reaction volume, the interaction probability P_s is defined as

$$P_S = k_b \left(\frac{s_s - d}{s_s} \right), \tag{3.7}$$

with k_b denoting the binding rate ([1/s]) of molecules to the surface. If a random number $\xi \sim \mathcal{U}(0, 1)$ is smaller than the interaction probability P_s, the particle is adsorbed onto the surface of the structure. The particle type is then adjusted accordingly to its new type. The position of the particle is also adjusted such that the particle touches the surface it is bound to. In case the particle is attached to a filament of the cytoskeleton, it becomes immobile and is no longer affected by diffusion. If a non-zero transport velocity is assigned to the interaction, the particle is moved by motorized transport.

The dissociation or detachment of the attached particle depends also on a threshold, the unbinding probability. In every subsequent time step, a random number $\xi \sim \mathcal{U}(0, 1)$ is drawn and compared against the threshold. The particle detaches immediately when the random number is smaller. Depending on the interaction rules, the original particle type is restored and the particle moves again according to Equation 3.1.

3.1.5 Transport by Motor Proteins

The motorized transport or transport by motor proteins uses different proteins to move the cargo along the filaments. On microtubules dynein and kinesin are used whereas myosin is used for the transportation along microfilaments [Lodish et al., 2007]. When a particle is attached to a cytoskeletal filament, its movement is constrained onto the filament direction simulating the piggyback transport by motor proteins. The movement of the actively transported particles, therefore, deviates from (3.1). Instead of a random step, a movement with a constant velocity following the normalized direction of the filament $\boldsymbol{d}_f$ is used, yielding

$$\boldsymbol{x}_i(t + \Delta t) = \boldsymbol{x}_i(t) + \boldsymbol{d}_f v_f \Delta t, \tag{3.8}$$

where v_f is the velocity of the respective motor proteins. A positive velocity results in a movement toward the negative end of the filament, i.e. toward the nucleus. The opposite direction of movement is obtained with a negative velocity. This allows the modeling of the protein transport along microtubules toward the nucleus with dynein and away from it with kinesin. In addition to the dissociation depending on probabilities (cf. Section 3.1.4), particles reaching one end of the filament also detach from it.

3.1.6 Reactions between Particles

Particles can react with each other in biochemical reactions where one or more reactants are converted into one or more products. The necessary conditions for such reactions include that the participating molecules are close enough together, both possess the correct orientation with respect to each other, and the reaction is energetically possible. The correct orientation is compulsory for protein-protein interactions to allow for bonds between the respective binding sites. The particles in our simulation, however, are spherical thereby neglecting the spatial characteristics of the represented molecules. Due to the fast rotational diffusion of a molecule, the binding sites of one molecule are equally distributed on the sphere's surface and are, therefore, equally probable to be in the correct orientation (cf. Section 3.1.3).

Instead of chemical reaction rate constants, the particle simulation employs probabilities to model biochemical reactions and interactions. Since these probabilities are not commonly used and are typically dependent on Δt, the macroscopic reaction rate constants given in $[\mathrm{L\,mol^{-1}\,s^{-1}}]$ have to be converted from the continuous models to probabilities used in the discrete model. The reaction probabilities allow to model reactions between particles without considering the global molecule concentrations. In case the particles are reacting, the species of the particles are adjusted in accordance with the reaction and their positions remain unchanged. For reactions with more products than reactants, or more reactants than products, particles have to be spawned or destroyed, respectively.

First-order Reactions

First-order reactions describe the spontaneous decay or state change of a molecule, for instance a deactivation or unbinding process. If a molecule reacts with a non-modeled entity, e.g., a hormone or a protein, this can also be considered to be a first-order reaction. The change in the concentration c_i of molecule i is proportional to the monomolecular reaction rate constant k_i

$$\frac{\mathrm{d}}{\mathrm{d}t}c_i = -k_i c_i. \tag{3.9}$$

Assuming a discrete time step Δt, Equation 3.9 can be discretized as follows

$$\frac{\Delta N_i}{\Delta t} = -k_i N_i, \tag{3.10}$$

where N_i denotes the number of molecules and ΔN_i the change during Δt. Reformulating Equation 3.10 yields the probability P_i with which a molecule of species i reacts within the time span Δt:

$$P_i = \frac{|\Delta N_i|}{N_i} = k_i \Delta t. \tag{3.11}$$

To consider the change of N_i within a time step Δt the probability has to be corrected according to Andrews and Bray [2004] yielding

$$P_i = 1 - \exp(-k_i \Delta t). \tag{3.12}$$

However, Δt has to be chosen in a way so that P_i is less than 1. Using the probability obtained in Equation 3.12, the first-order reactions of single particles can be carried out in the simulation. For every particle of species i the probability is compared against a random number $\xi \sim \mathcal{U}(0, 1)$. If the random number is smaller, the reaction will take place. While the original particle changes its type to match the species of the first product, a new particle has to be created for every additional product of the reaction. These new particles are placed at the position of the existing particle, because any other position might be blocked by cellular structures.

Reversible reactions like A $\rightleftharpoons$ B + B could lead to an immediate re-association of the dimer A after its decay, because the newly created monomer particles B are in contact since they share the same position. Therefore, newly created particles should not react in the time step after their creation. Andrews et al. [2010] artificially separate the products in space using an unbinding radius in order to prevent the immediate re-association. This implies, however, that two valid particle positions are found without intersecting with any cellular structure, i.e. additional collision tests with the cellular structures are necessary. In contrast to the approach of Andrews et al. [2010], a newly introduced *reaction delay* is used in the proposed model to ensure that immediate follow-up reactions are prevented.

The reaction delay τ is implemented as timer for each particle that is decreased by Δt in the subsequent time steps. As soon as τ is zero, the particle is again allowed to participate in reactions. Since Δt is only on the order of microseconds, the introduced error due to

the waiting time is negligible for the simulated signaling process, which covers seconds to minutes. In general, the delay gives the particles enough time to diffuse away from each other. Depending on the diffusion coefficient this escape process might, however, become diffusion-limited. The reaction delay also allows one to model recuperation processes of more complex structures without adding additional molecules and reactions to the model and therefore can be extended to simulate enzymatic reactions.

Second-order Reactions

Second-order reactions describe reactions according to mass action kinetics between two molecules, i.e. a bimolecular reaction. The two reacting particles have to be close enough together to facilitate the reaction. Higher-order reactions, i.e. third order and higher, are decomposed into a series of delayed second-order reactions because the probability that the particles meet each other in the three-dimensional domain decreases drastically. In addition, from a biological perspective higher-order reactions are often formed by a number of subsequent reactions.

Particle-particle interactions are based on their distance to each other. Pogson et al. [2006] use a maximal reaction distance, i.e. the maximal possible distance between the reacting particles i and j, to define a reaction volume around one particle. If the second particle enters this reaction volume, both particles might react. This approach was extended by Klann et al. [2011] to cover effects caused by the diffusion limit. The reaction probability given by

$$P_{ij} = \frac{\kappa_{ij}\Delta t}{\frac{4}{3}\pi(r_i + r_j)^3}, \tag{3.13}$$

is derived from the microscopic reaction rate constant κ_{ij}, with r_i and r_j denoting the respective particle radii [Klann et al., 2011]. The macroscopic reaction rate constant k_{ij} and the diffusion limit γ_{ij} can be used to calculate κ_{ij}

$$\frac{1}{\kappa_{ij}} = \frac{1}{k_{ij}} - \frac{1}{\gamma_{ij}\mathrm{N_A}} \quad \text{with} \quad \gamma_{ij} = 4\pi(r_i + r_j)(D_i + D_j), \tag{3.14}$$

where $\mathrm{N_A}$ is Avogadro's constant and D_i, D_j are the diffusion coefficients of the particles i and j, respectively. Given Equation 3.13, the reaction between two particles can be described with the following algorithm:

if $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| \leq (r_i + r_j)$ ***and*** $\xi \sim \mathcal{U}(0,1) < P_{ij}$ **then**
  particles i and j react;
end

Thus, if the distance between two particles is smaller than their combined radius and a random number $\xi \sim \mathcal{U}(0,1)$ is smaller than the reaction probability P_{ij} the reaction takes place. Again, care has to be taken, that all probabilities P_{ij} are less than one. Otherwise Δt has to be adjusted for correct results.

Enzymatic Reactions

The third class of reactions is a special case of the second-order reactions. In enzymatic reactions, an enzyme catalyzes an reactant and is both, reactant and product. The enzyme E converts the substrate S into the product P by forming an intermediary complex ES during this process: $\mathrm{E} + \mathrm{S} \rightleftharpoons \mathrm{ES} \longrightarrow \mathrm{E} + \mathrm{P}$. Besides the simple mass action kinetics, biologists also use Michaelis-Menten kinetics [Michaelis and Menten, 1913] to describe such enzymatic reactions. The set of three reactions can be described with only two parameters, namely the apparent unimolecular rate constant k_{cat} and the Michaelis constant $\mathrm{K_m}$. The unimolecular rate constant k_{cat}, the so-called turnover number, determines the maximum number of substrate molecules which can be converted by a single enzyme per second. The enzymatic reaction is simplified to $\mathrm{E} + \mathrm{S} \longrightarrow \mathrm{E} + \mathrm{P}$ omitting the intermediary complex ES. The product formation rate r_p is then given by

$$r_p = \frac{k_{\mathrm{cat}}}{\mathrm{K_m} + c_{\mathrm{S}}} c_{\mathrm{E}} c_{\mathrm{S}}, \tag{3.15}$$

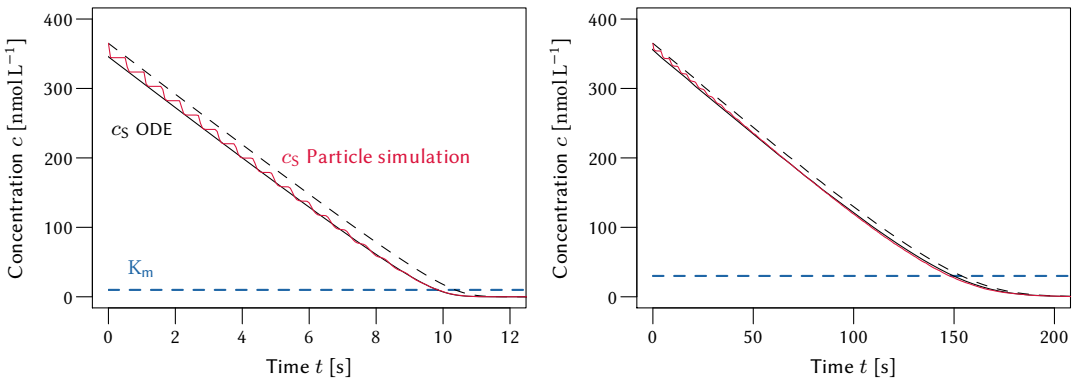
with c_{X} being the concentration of compound X. Equation 3.15 is basically a second-order reaction, where the rate constant is modified by the concentration of the substrate S, i.e. $k_{\mathrm{ES}} = \frac{k_{\mathrm{cat}}}{\mathrm{K_m} + c_{\mathrm{S}}}$. Since a particle-based simulation is used, the local concentration c_{S} is not readily available.

Applying queuing theory allows one to model the Michaelis-Menten kinetics in the stochastic simulation without computing concentrations [Smith, 1971]. The reaction should happen with the maximal possible rate $k_{\mathrm{ES,max}}$ that is obtained when the substrate concentration c_{S} vanishes, i.e. $c_{\mathrm{S}} \to 0$ and, therefore, $k_{\mathrm{ES,max}} = \frac{k_{\mathrm{cat}}}{\mathrm{K_m}}$. A reaction delay $\tau_{\mathrm{E}} = \frac{1}{k_{\mathrm{cat}}}$ is used to prevent follow-up reactions of the enzyme E in order to account for the saturation of the reaction at higher substrate concentrations c_{S}.

The errors introduced by this approach are not bigger than the deviations between the Michaelis-Menten kinetics and the original reaction scheme with three reactions, given that the underlying quasi-steady-state assumption holds. The enzyme concentration has to be low and the Michaelis constant high to fulfill the assumption, i.e. $c_{\mathrm{E}}(0)/(c_{\mathrm{S}}(0) + \mathrm{K_m}) \ll 1$ [Segel and Slemrod, 1989]. Figure 3.3(a) shows that the reaction delay is visible as steps in the substrate dynamics for improper parameter values ($c_{\mathrm{E}} > \mathrm{K_m}$) and the particle simulation, therefore, deviates from the ODE solution. If the quasi-steady-state assumption holds, the simulation results are in accordance with the results of an ODE model as depicted in Figure 3.3(b). Since all enzymes are initially active, the stochastic simulation shows an initial burst phase that shifts the curve.

3.1.7 Transport through Nuclear Pores

The nuclear import and export of signaling molecules determines the efficiency of the signal transduction process from the plasma membrane to gene regulation [Fujioka et al., 2006]. The nuclear envelope has to be passed in order to exchange molecules between the

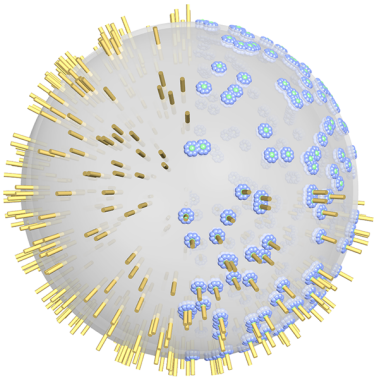


▲ **Figure 3.3 —** Analytical and numerical solution of an enzymatic reaction with applied Michaelis-Menten kinetics for the substrate concentration c_S. The results of the particle simulation (red) are compared to the ODE solution (black). (a) improper parameter set with $c_E > K_m$. (b) quasi-steady-state assumption holds, i.e. $c_E \ll K_m$ (parameters are given in Appendix A.2).

cytoplasm and the nucleus. The exchange happens through the nuclear pore complexes, which are distributed on the outer nuclear envelope. The individual complexes allow certain molecules to diffuse through the pore and actively transport larger molecules from one side to the other. In the proposed cellular model, the import and export through the nuclear pore complexes is handled in the same way as the transport by motor proteins along microtubules or microfilaments.

In the model, the nuclear pores are assumed to be cylindrical holes that extend into the cytoplasm and into the nucleus. The cylindrical representation of the nuclear pores used in the simulation is shown on the left in Figure 3.4. For the visualization during the analysis step, the cylinders are replaced by an approximation of the nuclear pore complex with nine spheres, eight on the outer perimeter and one in the center.

The length of the cylindrical representative directly correlates with the probability with which a particle binds to a nuclear pore and can, therefore, be used to adjust the binding rate to match the respective reaction rates obtained in experiments. Similar to the motorized transport, molecules attached to nuclear pores are not subject to diffusion but linear movement. Instead of moving on the surface of the cylindrical representation of microtubules, the particles are positioned on the central axis of the cylinder. The particle positions are updated according to Equation 3.8 moving along the cylinder through the pore. In contrast to the motorized transport, particles transported through nuclear pores are not allowed to detach from the pore unless they have reached the end of the cylindrical pore.



▶ **Figure 3.4** — Modeling nuclear pores. During the simulation, the nuclear pores are handled as special filaments (left part). The nuclear pore complex is represented by several spheres in the visualization (right part).

3.1.8 External Stimuli

Signal transduction processes are triggered by external, extracellular effects. Proteins and hormones moving through extracellular space stimulate membrane-bound receptors and induce an intracellular signal. When the external signal is not present on the complete membrane, the result is a polarized cell, an effect which typically occurs in tissue or blood vessels. Such polarization effects are also found during the chemotaxis of cells [Slaughter et al., 2009]. The simulation of these effects enables for comparison with wet lab experiments, e.g., using microfluidic devices. Microfluidics allows the control and manipulation of flows on the microscopic scale. In cell biology, microfluidic devices are used to establish and control a defined extracellular environment where living cells are located in. Small reaction chambers are etched on a microchip and are connected with complex tubing to one or more nutrition feeds. The tubing allows adjusting different concentration scenarios inside the reaction chamber including concentration gradients or concentrations changing over time and thereby enabling the analysis of the effects on cells.

Since the extracellular space is not included in the simulation model, the extracellular concentrations are modeled with the help of mathematical functions dependent on both time and space. In addition to the particle attributes described above, an activation state is added to each particle (cf. Section 3.1.1). The activation state of all particles that are not affected by the external signal is initially set to active and remains unchanged during the simulation. In the following, only particles are considered whose activation state might change, i.e. those particles affected by the external effects. Assuming a steady state, the probability P_0 that a molecule is initially active at $t = 0$ is given by

$$P_0 = \frac{k_+}{k_+ + k_-}, \tag{3.16}$$

where k_+ and k_- are the rate constants of activation and deactivation, respectively. If a random number $\xi \sim \mathcal{U}(0,1)$ is smaller than P_0, the activation state is set to active.

During the simulation, the activation state of every particle is updated at each time step taking into account the particle's position $\boldsymbol{p}$ at time t. The prevalent external signal concentration at position $\boldsymbol{p}$ is defined by the stimulus function $f(t, \boldsymbol{p})$. The function

is limited to three different forms without the loss of generality. In case of a constant concentration outside the cell, the stimulus function simply evaluates to the predefined threshold c_{min}, i.e.

$$f(t, \boldsymbol{p}) = c_{\text{min}}. \tag{3.17}$$

A concentration gradient is modeled by a linear interpolation between two points $\boldsymbol{a}$ and $\boldsymbol{b}$ in $\mathbb{R}^3$. The particle position $\boldsymbol{p}_i$ is projected onto a given line $\boldsymbol{ab}$ and the corresponding concentration is obtained by

$$f(t, \boldsymbol{p}) = (1 - \lambda)c_a + \lambda c_b \quad \text{with } \lambda = \frac{\langle \boldsymbol{p}_i - \boldsymbol{a}, \boldsymbol{b} - \boldsymbol{a} \rangle}{\|\boldsymbol{b} - \boldsymbol{a}\|} \tag{3.18}$$

with λ being the parameter of the line $\boldsymbol{ab}$ and c_a and c_b being the concentrations at $\boldsymbol{a}$ and $\boldsymbol{b}$. The third activation function describes a signaling pulse by a box function. Let d be the distance between the origin of the pulse $\boldsymbol{p}_o$ and the particle position, then the distance d', due to periodicity of the pulse, can be defined as

$$d' = d - \left\lfloor \frac{d}{d_p} \right\rfloor d_p \quad \text{with } d = \|\boldsymbol{p}_o - \boldsymbol{p}_i\| \tag{3.19}$$

where d_p is the distance between two pulses. Using the modified distance d', the stimulus function is then given as

$$f(t, \boldsymbol{p}) = \begin{cases} c_{\text{high}} & \text{if } (t + t_0)v < d' \leq (t + t_0)v + w \\ c_{\text{low}} & \text{otherwise,} \end{cases} \tag{3.20}$$

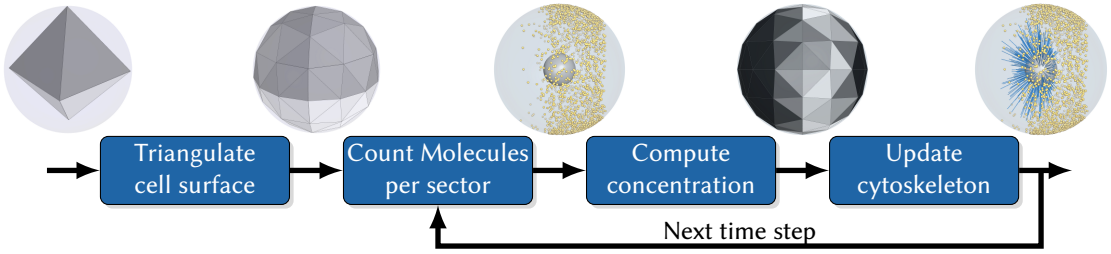
where t_0 is the time where the pulse begins, w is the width of the pulse, and v the velocity along the direction of propagation. The signal concentration of the pulse is given by c_{high}, whereas the concentration of the remaining parts is c_{low}. The stimulus function f and the respective reaction rates k yield the probability of a state change of particle i:

$$\begin{aligned} P_i(t) &= k_+ f(t, \boldsymbol{p}_i)\Delta t \quad \text{(activation),} \\ P_i(t) &= k_- f(t, \boldsymbol{p}_i)\Delta t \quad \text{(deactivation).} \end{aligned} \tag{3.21}$$

The activation state of the particle only changes if a random number $\xi \sim \mathcal{U}(0, 1)$ is smaller than $P(t)$. By using this stochastic approach, the resulting changes in state are no longer purely analytical but reflect a more natural, random behavior.

3.1.9 Dynamic Cytoskeletal Filaments

In the previous section, effects that extracellular signals have on the activation of molecules were described. However, the structure of the cell, including the plasma membrane and the cytoskeleton, changes also dynamically with respect to external signals. These changes enable cells to detect salient directions of interest and to move toward the source of the signal [Mogilner et al., 2012].

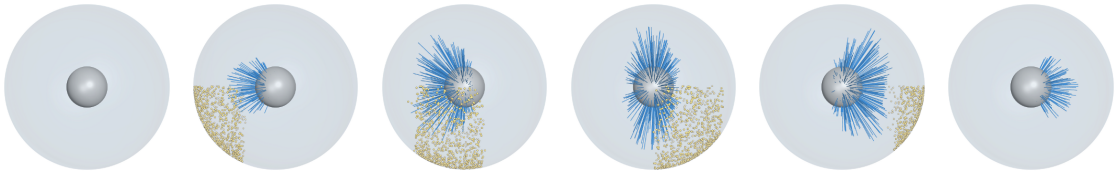


▲ **Figure 3.5** — Algorithm for updating the dynamic cytoskeleton. The surface of the plasma membrane is approximated with a triangle mesh to obtain averaged local concentrations. These concentrations are used to affect the growth of the cytoskeletal filaments.

Microfilaments and microtubules can grow and shrink dynamically [Guo et al., 2010]. Signaling complexes can control this process and thus generate filaments that lead into the direction of the signal [Li et al., 1995; Slaughter et al., 2009; Mogilner et al., 2012]. In the cell, the filaments grow by a polymerization of monomers at one side, i.e. α and β tubulin for microtubules and actin for microfilaments. The modeling of the process on this scale requires to track all the monomers and to identify the values of the kinetic binding and dissociation rate constants [Guo et al., 2009]. Instead of this fine-grain approach, the length of the cylindrical representatives of the filaments is adjusted without modeling the monomers. The length is determined by the amount of activating molecules in the corresponding plasma membrane domain. Local average concentrations of the respective molecular species, which controls the length of the filaments, are estimated to avoid the computation of concentrations for every filament. Since the signaling complexes are typically located at the plasma membrane the surface concentrations of those complexes is used instead of volume concentrations. In a four step process, the local surface concentrations are determined and the lengths of the cytoskeletal filaments are adjusted accordingly. The necessary steps of the algorithm are depicted in Figure 3.5.

First, the spherical plasma membrane is approximated with a triangle mesh. Beginning with an octahedron, the sphere approximation is obtained by a subdivision of the triangle faces. Two or three levels of subdivision are sufficient for the approximation of local concentrations because the average surface area of the subdivided triangles would become too small at higher subdivision levels resulting in local concentration peaks.

In the next step, the number of active particles is determined for every triangle of the sphere approximation. All particles lying in the tetrahedral space spanned between a triangle and the cell center, i.e. one sector, are considered for that triangle. Given the number of particles per sector and the triangle's surface, the surface concentration can be computed. The computed concentration is then used to update the dynamic elements of the cytoskeleton. The growth of the filaments is influenced by the surface concentration of the sector, where the current end point of the filament is located in. If the surface concentration is less than c_{shrink}, the filaments are decaying at the maximum possible rate with the catastrophic velocity v_{catastr}. Bounded by the minimal and maximal concentrations c_{shrink} and c_{grow}, respectively, the growth velocity v is determined by linear interpolation



(a) $t = 0\,\mathrm{s}$ (b) $t = 0.08\,\mathrm{s}$ (c) $t = 0.16\,\mathrm{s}$ (d) $t = 0.24\,\mathrm{s}$ (e) $t = 0.32\,\mathrm{s}$ (f) $t = 0.4\,\mathrm{s}$

▲ **Figure 3.6 —** Dynamic cytoskeleton. A single, external signal pulse sweeps over the cell and activates receptors. The microtubules (blue) grow depending on the receptor concentration thereby following the signal pulse. The activated receptors (yellow) are only shown in the lower half (parameters are given in Appendix A.7).

between v_{shrink} and v_{grow}. The length of the filaments is then adjusted by the growth $l = v\Delta t$ in the time step. The last three steps of the algorithm are repeated for each subsequent time step. Figure 3.6 shows the results of a single pulse affecting radial microtubules over time. When the surface concentration is high enough, the microtubules begin to grow. The growth comes to a stop and is reverted as soon as the concentration drops. After a short time, the shrinkage of the filaments is taking place with the maximal catastrophic velocity leading to an almost immediate decay clearly visible in the image sequence.

3.2 Graphical Notation for Describing Intracellular Processes

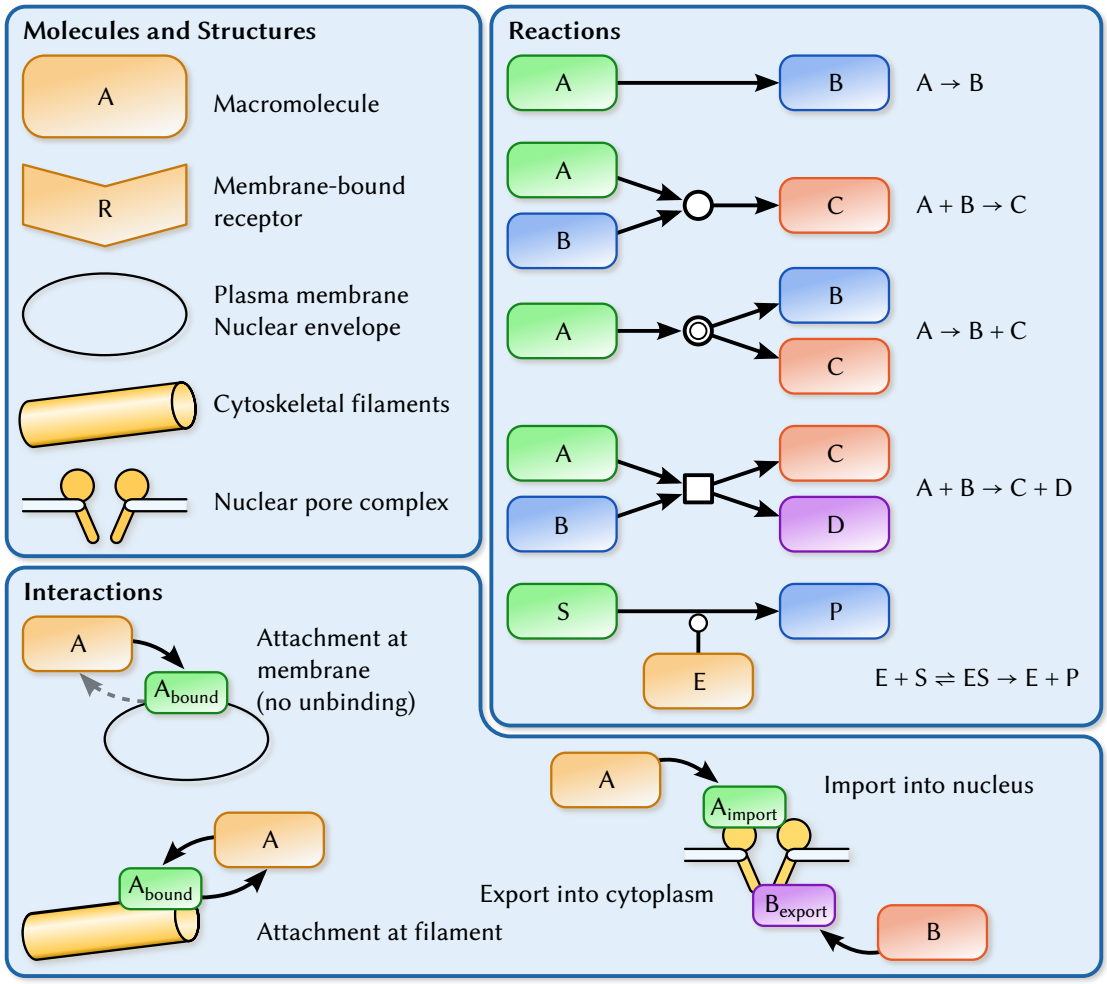
Biochemical reaction networks can be defined in several ways. Typically, such networks are either described as graphs or with the help of a text-based representation. Text-based methods have the advantage of a precise and powerful expressiveness. This fact is exploited in rule-based modeling, for instance with the kappa language [Danos et al., 2008] or BioNetGen [Blinov et al., 2004; Faeder et al., 2005], where, e.g., the phosphorylation states of a protein can be expressed without explicitly modeling every state (two phosphorylation sites could lead to eight different states). Smith et al. [2011] use a text-based representation in combination with a visual representation to allow for debugging of the model.

Using graphical representations to model biological networks has been investigated before [e.g. Kohn, 1999; Pirson et al., 2000; Kitano, 2003]. Compared to mere text-based descriptions a graphical display simplifies overall usability. However, care has to be taken when using arrows indicating connections. They can be ambiguous or have different meanings dependent on the context and, of course, the reader's knowledge. Kitano et al. [2005], therefore, propose a set of symbols for the systems biology mark-up language (SBML) mentioning that issues in human-readable representations were mostly ignored before. The systems biology graphical notation (SBGN) [Le Novére et al., 2009] is the most recent attempt to standardize the notation of biological processes. After expressing the chemical interactions between molecules, the graphical representation can

be converted into SBML and then simulated, e.g., with a system of ODEs in CellDesigner [Funahashi et al., 2008]. The SBGN provides containers to model different compartments but does not consider the spatial context that is of some importance in cells. It also focuses only on chemical reactions between molecules and neglects the modeling of cellular structures like the cytoskeleton and the interactions with those structures.

In the proposed notation, the sketched model should roughly reflect the location of the individual entities during the simulation and should include all possible interactions in an easily understandable manner. For instance, molecules can be located only in the cytosol or inside the nuclear envelope and membrane-bound proteins are attached to the respective membrane. In addition to the spatial context, interactions between molecules and structures are explicitly modeled and depicted. The notation is loosely based on the SBGN and reuses the same shapes for molecules and chemical reactions. In Figure 3.7, the entities for modeling molecules and structures are depicted as well as different reaction types and possible interactions between structures and molecules. Each shape used in a model is a representative of the respective species, i.e. for each molecular species only one instance has to exist in the model. As described above, the environment of the cellular model comprises at least the plasma membrane and the nuclear envelope of the nucleus. The membrane-bound receptors are a special kind of molecules that are attached to the plasma membrane at all times during the simulation. Regular molecules can temporarily become attached but they might detach again and can be placed inside the plasma membrane like the cytoskeleton representatives. Reactions between molecules are indicated by straight one-way arrows. Depending on the number of reactants and products, the reaction symbol changes, thereby indicating the reaction type. Enzymatic reactions of the form $E + S \rightleftharpoons ES \longrightarrow E + P$ are shown as a conversion of the substrate S to the product P catalyzed by the enzyme E. Curved arrows mark possible interactions between a molecule of a certain species and the respective cellular structure. In case of the interaction with the cytoskeleton or a membrane, i.e. either plasma membrane or nuclear envelope, two curved arrows connect the molecule with a second molecule representing the molecule in the attached state. Dashed arrows are used to indicate binding rates of zero, if the molecule is not allowed to attach or unbind from the structure. This approach is adapted for the nuclear import and export by the core pores. Since the unbinding of molecules is not allowed for molecules in transit, only one arrow directed from the original molecule to the one transported through the pore is shown.

The notation given in Figure 3.7 was used in a prototypical implementation supporting the users in the first stage of model development. A graph-based editor is used to design and modify new models interactively. The tool allows modeling cellular interaction networks by placing molecules and cellular structures and adding connections. The necessary parameters used in the simulation and additional settings for the subsequent analysis, like, e.g., colors of molecular species, are all defined within the editor. The model is stored in an XML format, which is also understood by the simulation and the analysis stage of the model development cycle.



▲ **Figure 3.7** — Graphical notation for describing cellular models. The notation is based on the SBGN and was extended to consider interactions between structures and molecules.

CHAPTER 4

PARTICLE-BASED CELLULAR SIMULATION

The cell model described in the previous chapter is evaluated with a parallel, stochastic MC simulation. The spherical plasma membrane of the cell confines the simulation domain. The cellular interior is modeled with a static nucleus and populated with cytoskeletal filaments and various dynamic signaling molecules, represented as spherical particles. The stochastic simulation determines the movement and the outcome of possible interactions between particles. The original simulation by Klann [2011] serves as basis for the parallel implementation.[1]

Different possibilities exist for the parallel execution of the simulation. Compute clusters, e.g., consist of many interconnected individual nodes with non-shared memory and isolated CPUs. They are best suited for medium-sized to large problems because of the necessary communication between the nodes and the need for load balancing. Cellular simulation, however, can be considered to be a small-sized problem requiring much more communication given the chemical interactions between the particles. Even in the case of several hundred thousand particles, the memory requirements are still low. Hence, today's many-core processors, CPUs as well as GPUs, are well-suited for massively parallelized particle simulations with their SIMD and SIMT architectures, respectively. The simulation is performed on a single compute node, but the computation is broken into several parts, each handled by a separate thread. The communication between threads still remains necessary but it occurs at much higher speed. The focus in this thesis is on a parallel GPU implementation with CUDA. In addition, the parallelization approaches are adapted to the CPU architecture as well resulting in a parallel, and also optimized, CPU solution of the simulation to allow for baseline measurements.

[1] **Parts of this chapter have been published in**: M. Falk, M. Ott, T. Ertl, M. Klann, and H. Koeppl. Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology. In *International Conference on Computational Methods in Systems Biology (CMSB 2011)*, pages 73–82, 2011.

First, the general problems when dealing with the parallelization of chemically interacting particles are described and solutions are presented. This is followed by the basic algorithm of the simulation and implementation considerations depending on the underlying hardware. The chapter concludes with an extensive evaluation covering the raw performance of the parallel implementations and corresponding application scenarios.

4.1 Parallelization Strategies and Problems

The peak performance of recent GPUs promises high speedups especially when compared with equivalent CPU implementations. Despite the impressive raw performance numbers, the feasibility is often overlooked of whether a particular algorithm can be parallelized and which components of it still have to be executed sequentially. In the context of particle simulations, this comes down to simple particle systems without any interactions yielding tremendous speedups when performed on the GPU.

The parallel implementation of the SSA [Li and Petzold, 2010], or the Gillespie algorithm, is such a case. Here, only one probability measure has to be determined once per simulation step and then all particles are processed independently, changing their type according to the previously calculated probability. The approach presented by Cecilia et al. [2010] makes use of the inherent parallel nature of membrane systems. Membranes separate different reaction spaces and can hence be computed in parallel. Inside each membrane, rules are selected and applied for every appearing species in a parallelized way. Lattice-based simulations, used to simulate reaction-diffusion systems, make use of the subdivision of the simulation domain and process the individual cells of the lattice in parallel. Care has to be taken for some special cases, like particles entering cells that are already full [Roberts et al., 2009]. All of the above mentioned approaches benefit from the massively parallel GPU architecture because their computations are mostly independent of each other. Interactions between individual threads, however, are reduced to a minimum, given the various requirements and restrictions on the realization of parallel simulations, including the need for simplification or the disregard of special cases.

As the cellular model proposed in this thesis incorporates structural and chemical interactions besides the diffusion of particles, two further challenges are identified due to the parallelization. Both have in common that the corresponding interaction partners have to be identified as quickly as possible. The first challenge is the interaction of particles with static cellular structures and requires the fast detection of collisions. The second one is concerned with second-order reactions. The algorithmic challenge here is the parallel handling of reactions between particles and their reaction products.

4.1.1 Structure Interactions

Simple particle diffusion according to Equation 3.1 can be parallelized easily, since the particles move independently. Collision detection between particles is neglected in favor

of the chemical reaction model and the parallelization (cf. Section 3.1.3). The particle diffusion, however, is affected by the interactions with static cellular structures, including membranes, cytoskeletal filaments, and other static obstacles. The efficient detection of collisions between a particle and nearby structures is the key to a fast and load-balanced, parallel computation.

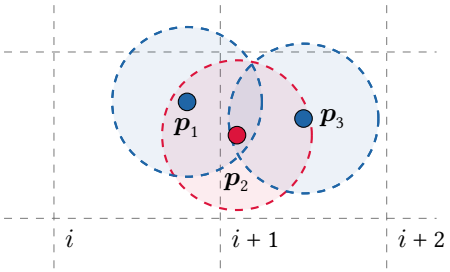
The number of structures m is low compared to the overall number of particles n, but testing all possible collisions is still in the order of $\mathcal{O}(mn)$. To reduce the complexity, various approaches for space partitioning can be employed, including uniform grids [Fujimoto et al., 1986], binary space partitioning [Kaplan, 1985], octrees [Glassner, 1984], or bounding volume hierarchies [Kay and Kajiya, 1986]. Cellular structures, especially the cytoskeletal filaments, are randomly distributed in the cytoplasm, limiting the advantages of hierarchical approaches because of the approximate uniform distribution. The GPU implementation of hierarchical approaches can exhibit another disadvantage—the traversal of the hierarchy. While the number of possible collision objects is reduced, the entire hierarchy has to be traversed for each single particle, which could lead to load-imbalances between threads. Assuming a uniform distribution of cellular structures, except for the membranes of course, and the demand for efficient access of the acceleration structure on the GPU, uniform grids are best suited. For moderate grid dimensions, the additional expenses in memory are marginal compared to the overall memory consumption of hierarchical approaches.

The uniform grid is fitted to encompass the entire plasma membrane. All cellular structures are inserted into the cells of the uniform grid with which they intersect. To check for potential collisions, all grid cells covered by a spherical particle have to be considered. The method can be further refined by enlarging the cellular structures first and then inserting them into the grid, leading to slightly higher memory consumption due to duplicates. The factor for the enlargement is given by the largest particle radius. Thus, only the contents of the cell in which the particle is actually located in has to be checked for potential collisions without considering the particle's spatial extent. The collision detection itself is reduced to the computation of the minimal distance between two geometric objects, i.e. the spherical particle and the either spherical or cylindrical structure. Depending on the existing interaction rules, the particle might also react with intervening structures, if the distance between both is less than the necessary binding distance (cf. Section 3.1.4).

4.1.2 Chemical Interactions

The dependencies between particles make the parallelization of particle-particle reactions challenging. The appropriate reaction partners must be identified, in particular for second-order reactions, and a consistent simulation state must be ensured, i.e. the same particle must not be accessed by two concurrent threads.

Assuming a serialized implementation, reactions are completed in a sequential manner without any concurrency or side effects. Each particle has to be checked with every other particle resulting in a quadratic complexity $\mathcal{O}(n^2)$ for n particles. Since the interactions

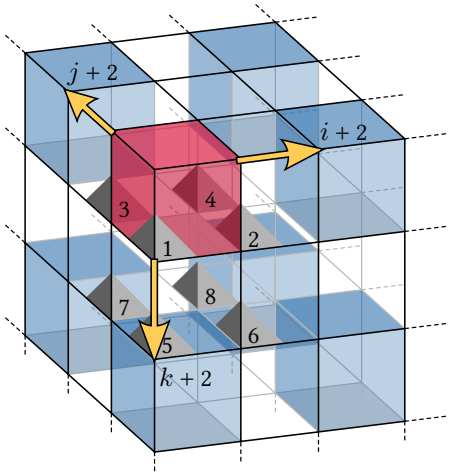


◀ **Figure 4.1** — Pitfalls in parallel handling of particle interactions. Due to their size and random position, particles can react with particles in different cells of the uniform grid. Two threads could simultaneously find the interactions between the particles $\boldsymbol{p}_1$–$\boldsymbol{p}_2$ and $\boldsymbol{p}_3$–$\boldsymbol{p}_2$.

between particles are solely based on the reaction distance, only the local neighborhood has to be searched for possible reaction partners. A uniform grid is used to restrict the neighborhood searches of the particles, similar to the one used for particle-structure interactions. Setting the edge length of the grid cells to at least twice the maximal occurring reaction distance between two arbitrary particles ensures that a particle can be part of reactions taking place in at most 8 neighboring cells. That is, once a particle is for example located in the left half of a cell, it cannot react with any particles located in the cell to the right of the current cell. Instead of visiting all 8 neighboring cells to find potential reaction partners, the original particles are duplicated and only the contents of one cell have to be tested. The particle is referenced up to eight times and inserted into the respective cells, which intersect with the reaction volume around the particle's position.

The computation of the chemical reactions can be either parallelized by considering each particle or by using the cells of the uniform grid. The parallelization over the individual particles requires the location of the grid cell for each particle and to test for all possible reaction pairs. This results in a complexity of $\mathcal{O}(nm)$, where n is the total number of particles and m the maximum occurring number of particles per grid cell. However, care has to be taken with reactions computed in parallel as the following example with the reaction A + B $\longrightarrow$ C will show. Assuming one particle A and one of species B, particle A might react with B in one thread and particle B could also react with A in a second thread at the same time. Thus, the assumptions of second-order reactions do not hold any longer as the reaction is now twice as likely to happen. Additionally, the result of the reaction is heavily depending on the implementation, which could result in zero, one, or even two particles of type C.

The complexity can be reduced when the parallelization is carried out over the uniform grid. With a moderate grid size the number of cells, e.g., $k = 32 \times 32 \times 32$, is less than the number of particles. In each thread, all pairs of particles are considered for potential reactions resulting in $\mathcal{O}(km^2)$. The parallelization over the grid cells, however, does not account for the fact that a particle near the cell boundaries might react simultaneously in two independent reactions with two particles located in different cells. It might also happen that the reaction between the same particles takes place more than once in different cells thus greatly affecting the reaction probability. Figure 4.1 illustrates this case. Assuming that the grid cells i and $i + 1$ are processed by independent threads, the particle at position $\boldsymbol{p}_2$ can potentially react with both particles $\boldsymbol{p}_1$ and $\boldsymbol{p}_3$ in the same time step leading to inconsistent data. Since the type of particle $\boldsymbol{p}_2$ might change due to the first reaction, the second reaction must be prevented.



▶ **Figure 4.2** — Parallel processing of the uniform grid. Using 8 passes allows checking all cells of the uniform grid in an interleaved pattern without inconsistencies (passes indicated by gray pyramids).

To prevent the occurrence of inconsistent results and to achieve maximum parallelism, the computation of the second-order reactions is separated into different passes. Since a particle can only react with other particles in the nearest eight cells, all non-neighboring cells are computed in the same pass. The passes are interleaved in a way such that two concurrently processed cells are not adjacent as illustrated in Figure 4.2. In three dimensions, 8 passes are required. The indices for each pass are precomputed during the initialization stage and stored in device memory. The indices are then used in each pass where the threads carry out the reactions for each other cell of the uniform grid. A reaction delay τ of at least Δt ensures that if a particle has reacted in one pass, it will not be used again in another pass of the same simulation step.

4.2 Algorithm

The stochastic, cellular simulation is divided into several, independently parallelized parts instead of being executed as one large monolithic task. This design decision was made to increase maintainability and to ease the addition of new features. Additionally, this approach allows for an efficient execution of the simulation when some cellular features like the dynamic cytoskeleton are disabled. As mentioned above, the simulation was developed for parallel architectures of current many-core processors and is hence suited for the operation on GPUs and CPUs. In both cases, the CPU takes on the role of the coordinator, while the computationally expensive functions are performed in parallel on respective hardware. The algorithm outlined here gives an overview of the host, i.e. the CPU-side, code of the simulation.

```
initialization;
while termination condition is not fulfilled do
    random number generation;
    if external signals are enabled then
        activation state update;
    end
    movement and collision detection;
    first-order reactions;
    second-order reactions;
    reaction delay update;
    if dynamic skeleton is enabled then
        dynamic skeleton update;
    end
    if frame should be written then
        data read-back;
    end
end
```

The simulation is started after the initialization. Random numbers, which are needed in the subsequent steps, are generated and stored in the first step. Particles are moved by a random walk step and collision detection is performed. Subsequently, first-order and second-order reactions are carried out and the reaction delays are updated. To be able to write the simulation results to permanent storage, the data has to be read-back from the GPU. The termination condition can be set to limit the number of simulated time steps or to terminate the simulation if the particle count of a specific molecular species exceeds a threshold. The simulation is terminated when the termination condition is fulfilled.

For the GPU variant, each CPU statement in the algorithm calls one or several device functions on the GPU. These device functions initiate the computation with parallel CUDA kernels on the graphics hardware. In the CPU-side implementation, the calls of device code are replaced with for loops parallelized with OpenMP, but the computation itself remains the same to a large extent. In the subsequent description of the algorithm's individual steps, minor distinctions between CPU and the GPU implementations are pointed out. Conceptual differences between both implementations are discussed in Section 4.3.

4.2.1 Initialization

After loading the simulation configuration from the XML description, the simulation process is initialized. The reactions are analyzed and the respective reaction rate constants are converted to reaction probabilities according to Equations 3.11 and 3.13. Enzymatic reactions are converted to second-order reactions. If not specified by the user, the time step Δt is calculated to ensure that all probabilities are less than one. Lookup-tables are generated for all types of reactions to allow a fast lookup of reactions to include all participating particles.

Memory is allocated to hold particle positions, type IDs, delays, and additional data used for first-order and second-order reactions. Specific type information like diffusion coefficients for each particle species are stored only once per species and are accessed with the respective type ID of the particles. The simulation domain is then filled with the various cellular structures. The plasma membrane defines the domain boundaries as it is the largest object enclosing all others. The nucleus is added and the nuclear pores, if enabled, are randomly placed on its surface. The filaments of the cytoskeleton are placed inside the plasma membrane as per placement rules and are clipped at the plasma membrane and the nucleus. The clipping is not essential for the simulation process but eases the analysis afterward when filaments do not protrude out of the cell. Then, acceleration structures for particle-structure interaction and second-order reactions are initialized (cf. Sections 4.1.1 and 4.1.2). Clipped filaments are inserted into the uniform grid for fast particle-structure interactions, where each grid cell contains a list of all intersecting elements. For second-order reactions only the grid indices of the eight independent passes are precomputed. The uniform grid itself is allocated but not filled, since it is filled and updated during the simulation. Subsequently, the particles are placed in the model by drawing uniformly distributed positions inside the spherical plasma membrane while preventing intersections with the cellular structures. The movement state of individual particles is set according to their placement, i.e. either membrane-bound or freely diffusing.

Initial Activation State

Besides the position and type ID, particle attributes also contain a flag indicating the activation state of the molecule. Only for active states, the particle is allowed to participate in reactions. The initial activation state of every particle that is affected by extracellular stimuli is determined by the probability P_0 as given in (3.16) and a random number $\xi \sim \mathcal{U}(0,1)$. If ξ is smaller than P_0, the particle is initially active. The remaining particles, which are unaffected by external signals, are always in their active state. If external stimuli are disabled globally for the duration of the simulation run, the activation state of all particles is initially set to active because the activation state will not change and all particles are expected to be able to participate in reactions.

Dynamic Cytoskeletal Filaments

The computation of local protein concentrations, which are necessary for the dynamic adjustment of the cytoskeleton, is done with the help of a mesh approximation of the plasma membrane (cf. Section 3.1.9). Subdividing the eight faces of an octahedron yields a triangle mesh that is computed and stored during the initialization stage. A unique ID is assigned to each triangle and the triangle surfaces are calculated to be used later on. The initial lengths of the dynamic filaments are set to a user-specified length. This could be the original length, which is also the maximum possible length of the dynamic filaments.

4.2.2 Random Number Generation

The random number generation is the first step of the actual simulation loop. The stochastic simulation needs a large amount of random numbers, especially for particle movement. A fast algorithm for the generation of random numbers is, therefore, essential. In the CUDA implementation, the GPU version of the *Xorshift* random number generator by Marsaglia [2003] is used. The generator is included in the *curand* library shipped with CUDA. Once per simulation step, a pool of random numbers is generated. For every particle, three values are generated for particle movement and one for each first-order reaction. Additionally, the probabilities for second-order reactions have to be stored. All numbers are stored as linear array in device memory on the GPU.

The CPU-bound simulation employs the Mersenne Twister random number generator [Matsumoto and Nishimura, 1998], which is included in the recent C++11 standard. On the CPU, random numbers are only generated when required during the evaluation of a reaction.

4.2.3 Activation State Update

At the current time t and the particle position $\boldsymbol{p}$, the stimulus function (Equations 3.17, 3.18, and 3.20) is evaluated for all particles. Particles, which are affected by the external signal, can change their state depending on the stimulus function and the probability $P_i(t)$ of a state change (cf. Section 3.1.8).

4.2.4 Movement and Collision Detection

Particles are in one of several movement states. A particle might be freely diffusing through the cytoplasm, it might be membrane-bound, or it is attached to a structure. The difference between the membrane-bound state and the attached state is subtle but nonetheless is of much importance. The movement of membrane-bound particles is still a random walk, which is confined to a specific membrane. Particles might attach themselves to any cellular structure, if the interaction is defined. In the attached state, however, the particle movement is more limited. On spherical structures a particle gets attached and there is no subsequent movement at all. Directed movement according to Equation 3.8, i.e. motorized transport or import through nuclear pores, is possible along cylindrical filaments and nuclear pores.

A random walk is used to model the Brownian motion of the particles at each time step. According to (3.1), the new position is obtained by adding a random vector $\boldsymbol{\xi} \sim \boldsymbol{\mathcal{U}}(-\sqrt{3} \cdot \mathbf{1}, \sqrt{3} \cdot \mathbf{1})$.This is done in parallel for each particle in a single CUDA kernel or parallelized over all particles in OpenMP. The position of the particle is only set to the new location if it is not colliding with the cytoskeleton. New, temporary locations of the particles are considered for collision detection with cellular structures. The uniform grid described in Section 4.1.1 is used for an efficient access of structures in the local

neighborhood. If the particle is located inside a filament, the new position is immediately rejected and the particle remains at its old location. However, if the motorized transport along filaments is enabled for the particle, the condition for attaching the particle onto the filament is checked prior to rejection. In case of a successful binding, the particle position is adjusted to touch the filament surface and movement is restricted along the filaments axis. The particle becomes unbounded with a certain probability in the subsequent time steps or when it reaches the end of the filament.

4.2.5 First-order Reactions

Since there are no dependencies for first-order reactions, the parallelization is effective for all particles. Particles decay, change their type, or spawn a new particle. Decaying particles are marked as free. Since memory cannot be allocated or released dynamically during kernel execution on the GPU, care has to be taken when particles are destroyed or created. A free store management tackles this problem. Apart from the main particle list, a second list is used to store the indices of unused particles.

In the OpenMP implementation, the active particles are first partitioned into used and unused particles and then compacted. The used particles are then equally distributed over OpenMP threads and reactions are performed in parallel since there are no inter-particle dependencies. The reactions take place with the probability P_i (cf. Equation 3.12). Each thread manages a temporary buffer for spawned particles. If the buffer is full, the newly created particles are written back to the global particle list reusing entries of the so far unused particles. Freed particles are simply marked as unused.

The GPU implementation of first-order reactions differs slightly and is similar to the implementation of Dematté [2010]. During the computation of first-order reactions, flags are used to indicate the particle states *unchanged*, *free*, and *spawned new particle*. The flags are used as keys in a subsequent sorting step with particle IDs as values employing a parallel sort provided by the CUDA thrust library. Counting the identical keys gives the number of freed and spawned particles. Afterward, freed particles are stored in the free store list and spawned particles are instantiated in another kernel, which updates the free store list. Instead of compacting the particle arrays in each step with prefix sums as done by Dematté [2010], an additional flag is used for each particle denoting its state. The state can either be active or unused thus eliminating the need for compaction. This flag is checked at all other steps of the simulation and determines whether the computations take place or are skipped. An advantage of this method is that unique particle IDs are not stored and created explicitly when spawning new particles. The array indices of the respective particles can be reused as ID instead because they are coherent throughout a simulation run. Uniqueness and coherency of particle IDs are important properties in subsequent visualization, e.g., to visualize particle trajectories.

4.2.6 Second-order Reactions

A uniform grid as described in Section 4.1.2 accelerates the search for potential reaction partners of a particle for second-order reactions. Particles are sorted into the uniform grid, which is subsequently used for the computation of the reactions. For building the uniform grid on the GPU, a list-based approach [Le Grand, 2007] is used. Each grid cell has a unique ID and holds a list of particles it contains. The size of the list is bound by eight times the number of particles, since the reaction volume of a single particle overlaps with at most eight cells at a time. For every particle, the cell IDs of those cells intersecting with the particle's reaction volume are stored. After sorting the cell IDs, the start index of each cell is determined with a parallel prefix sum by respectively counting the elements referencing the same grid cell. While in the algorithm of Le Grand [2007] the list of start indices is further compacted to contain only non-empty cells, this compaction step is not considered in this thesis. The start indices are then used in the CUDA kernel performing the reactions.

In contrast to the GPU solution, the acceleration grid is filled sequentially on the CPU. All particles are scattered onto the grid and their IDs are stored in the respective grid cells. By using this sequential approach, the sorting and the parallel prefix sum operations as performed on the GPU are not required.

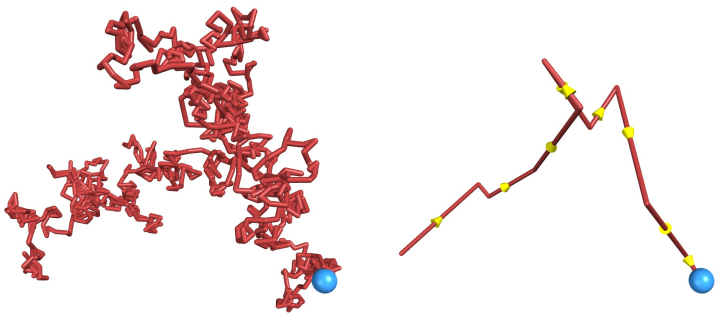
Once the grid is updated, the parallelization takes place over eight passes covering one eighth the total number of grid cells each. In every grid cell to be computed, the possible reactions between all particle pairs are checked and the respective reaction probability, as given in (3.13), is used. In case of a successful reaction, the reaction delay τ is set except for particle species in enzymatic reactions with $\tau_E = \frac{1}{k_{cat}}$, to prevent follow-up reactions of the reacted particles. Spawned particles are stored in a temporary list per grid cell and inserted into the global particle list after all computations of the current reaction pass have finished.

4.2.7 Reaction Delay Update

To prevent immediate reactions with newly created particles, an individual reaction delay τ is bound to each particle. The reaction delay has to be adjusted once per simulation step for each particle, i.e. $\tau' = \tau - \Delta t$. Particles with a delay greater than zero are declared inert.

4.2.8 Dynamic Skeleton Update

The algorithm for updating the dynamic elements of the cytoskeleton exhibits high similarity to the construction of the acceleration grid for second-order reactions. Instead of using IDs of grid cells, the unique triangle IDs of the mesh, which approximates the plasma membrane, are used on the GPU. For each contributing particle, i.e. a particle that affects the growth of the cytoskeleton, the corresponding triangle is determined and the respective triangle ID is assigned to the particle. All assigned IDs are sorted and



(a) (b)

▶ **Figure 4.3** — Details of the trajectories depend on the read-back frequency in the simulation. (a) every time step stored. (b) every 100th step stored. Simulation of 1000 steps with Δt = 75 ms and $D = 2\,\mu m^2\,s^{-1}$.

the number of identical keys is obtained with a parallel prefix sum over all IDs yielding the particle number per triangle. On the CPU, each particle is sequentially added to the respective triangle omitting the sorting step and the prefix sum.

Afterward, the locally approximated molecule concentrations are computed using the number of particles per triangle and thus the dynamic lengths of the cytoskeletal filaments can be adjusted. All filaments whose current end points lie in the triangular pyramid spanned by a triangle and the center of the cell are affected by the local concentration of the respective triangle. Depending on the concentration level relevant filaments grow or shrink (cf. Section 3.1.9). The dynamic lengths of the filaments are adjusted accordingly and are reused in the subsequent simulation steps.

4.2.9 Data Read-Back

The read-back frequency of the simulation results is defined by the user and affects the granularity of the data used in the subsequent analysis. For example, on longer time scales covering several seconds it might not be advisable to store every time step but only every hundredth or thousandth step. On the other hand, if the user is interested in the exact particle trajectories, every step has to be used and stored to permanent storage. The difference is shown in Figure 4.3 for a particle trajectory. Details of the Brownian motion are visible (Figure 4.3(a), every step stored) or an overview of the general direction of motion is given (Figure 4.3(b), every 100th step stored). However, a simulation can comprise millions of steps while a user might only be interested in the final molecule distribution.

The term read back is used in both contexts of GPU and CPU likewise to indicate that the intermediate data originating from the simulation has to be processed first before being written to permanent storage. Before the results of the current simulation step from the GPU are written to disk, they have to be copied from device memory of the GPU into main memory. In a small postprocessing step, the particle counts of all species are updated and unused particles are discarded. Later, the data of the remaining particles is written onto disk. During data read-back, the simulation state is not altered and the simulation continues with the consecutive steps after storing the data except that the termination condition holds.

4.3 Considering the Underlying Hardware

The parallel algorithm described above was developed with the aim of being portable on various many-core architectures. Depending on the underlying hardware, different design decisions are nonetheless necessary due to memory layout, memory management, and different architectural limitations.

The GPU is set up during the initialization stage of the simulation. Static data is stored in constant memory or in textures on the CUDA device. Texture fetches on the GPU are read-only as is the access of constant memory, but texture fetches are cached, which is beneficial when the textures themselves are small and the results are often used. Arrays containing information on reaction probabilities, reactants, and products as well as some of the particle data are, therefore, stored as textures. Constant memory is used for simulation parameters and pointers. To fully utilize the available memory bandwidth of the GPU, the memory layout of the particles' position data is optimized. Particle positions are swizzled for better memory access from the original layout $xyzxyzxyz\ldots$ to $xxx\ldots yyy\ldots zzz\ldots$. Before the position data is read back to the CPU it has to be reconverted first. Swizzling and unswizzling is both performed on the GPU within an additional CUDA kernel, requiring only an extra 60 ms for ten million particle positions.[2]

The parallel implementation of the simulation with OpenMP is straightforward, but also requires some additional considerations. The main difference between the GPU implementation and the CPU implementation with OpenMP is the partitioning of the computation into the individual threads, which are completely handled by OpenMP. Current CPUs typically feature fewer parallel cores than GPUs thus requiring theoretically less communication. Additionally, the overhead for switching contexts between threads is much higher. On the CPU-side, memory access and management is, however, easier. In particular, new memory can be allocated if needed for freshly spawned particles resulting from ensuing chemical reactions. On the GPU, more elaborate mechanisms like a custom-tailored free-store management are necessary. First-order and second-order reactions are handled slightly differently on the CPU, but use a similar free store management. Since memory management is not as limited as on the GPU, the handling of spawned or freed particles is more flexible. For example, if more reactions happen in one grid cell than the temporary buffer can hold, the contents of the buffer are written back instantaneously in sync with all other OpenMP threads. Such an operation would be too expensive when performed in CUDA kernels.

4.4 Performance Evaluation

The evaluation of the parallelized simulation is split into two parts. The first part deals with the overall performance of the simulation regarding the number of particles, the scalability of the simulation algorithm, and the resolution of the acceleration structures,

[2] Measured with an NVIDIA GeForce 670 GTX, 2 GiB RAM.

i.e. the size of the uniform grids. In addition, the results between the CPU and the GPU implementation are compared and discussed. The second part covers a qualitative analysis of different scenarios and can be found in Section 4.5. To perform the benchmarks a Intel Core i7 Dual Core in combination with a NVIDIA GeForce 670 GTX GPU was used (for details see Appendix B). The differences in the two operating systems Linux and Windows were negligible and, therefore, only the results acquired under Linux are presented in the following.

4.4.1 Scalability

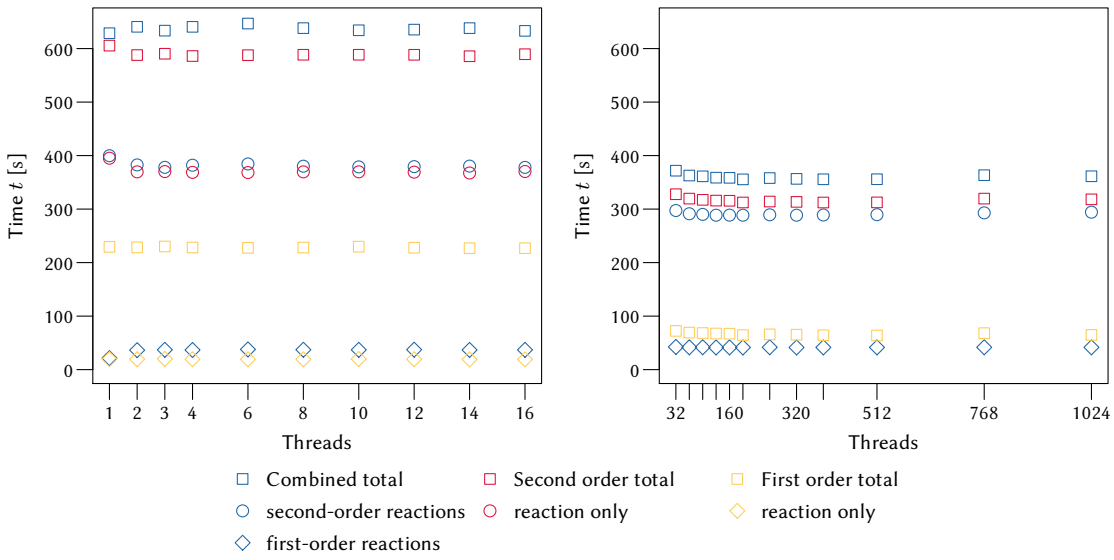
The scalability of the parallel implementation of an algorithm is an important factor for the general applicability of the algorithm. The algorithm might be optimal for a certain problem size, i.e. a certain number of particles, but might not be any more if the number of particles changes. Additionally, doubling the available computational power does not necessarily lead to doubling the performance. Scalability depends on the problem complexity and its size as well as the number of available processor cores or threads. Two types of scalability are regularly used to describe the behavior of an parallel algorithm in high performance computing: *strong* and *weak scaling*.

Strong scaling characterizes the effect on the overall computation time when the number of threads is varied and the size of the global problem remains fixed. As the problem size is constant, the workload per thread is ideally correlating with the total number of threads. Weak scaling on the other hand describes the algorithm's performance, if the problem size per thread remains the same, independent of the number of used threads. Hence, the global problem size has to be increased when the number of threads is incremented. Or if the number of threads is reduced, the overall problem has to shrink as well.

The simulation configuration used to demonstrate both types of scaling consists of five types of particles and two reactions, one of first order ($D \longrightarrow E$) and one of second order ($A + B \longrightarrow A + C$). The reactions are designed to maintain a constant number of particles in the scene during the simulation. Detailed parameter settings are necessary for the simulation setup and are given in Appendix A.3. In addition to strong and weak scaling, further measurements were gathered considering the performance of the simulation depending on the number of particles and the dimensions of the uniform acceleration grids. All measurements include only the time needed to perform the computation without any data read-backs.

Strong Scaling

The cell model was set up with two reaction types to analyze whether the simulation features strong scalability. The diffusion of the particles is hindered by filaments of the cytoskeleton, which occupy about 1.5 % of the cytoplasm volume. A baseline configuration of 50 000 particles was used throughout the benchmark (cf. Appendix A.3). On a typical CPU, the number of threads is varied between single thread and 16 threads, i.e. twice the

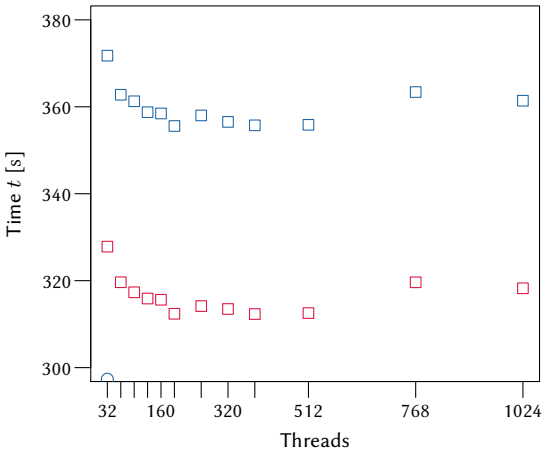


▲ **Figure 4.4** — Strong scaling performance test: OpenMP (left) and CUDA (right). The *reaction only* measurements of the test runs in CUDA with either first-order or second-order reactions are identical with the measurements of the combined test run and are, hence, not included in the plot to the right. Parameters are given in Appendix A.3.

number of available cores. For CUDA, the number of concurrent threads is set to be a multiple of 32, the number of threads executed in parallel, up to the maximum permitted number of threads per block, i.e. 1024.[3]

The problem size, i.e. the number of particles, remains constant over the simulation runs with different numbers of threads. In addition, the two reactions, one of first order and one of second order, are enabled and disabled alternately. This results in a total of three different simulation runs for each thread setting. Both reactions are enabled in the first run, while in the other two runs only one reaction type is enabled, i.e. either first- or second-order reactions. Detailed measurements were taken at each single step including the overall total time, time spent on the diffusion, and time spent for the first-order and second-order reactions. In addition, the time needed for the generation of random values was recorded on the GPU. Figure 4.4 shows the results for 50 000 steps of the OpenMP simulation running on the CPU (left) and the CUDA simulation (right). The performance difference between the reactions of first and second order shows up as expected in both plots. Since particles undergoing a first-order reaction have no external dependencies, the computation takes less time than for the second-order reactions, where potential reaction partners have to be identified. In the OpenMP simulation, first-order reactions take about 5.6 % of the total time and reactions of second order require 60 %. The remaining time is spent on the random walk, i.e. particle diffusion. On the GPU the ratios of the

[3] Both parameters are limited by the underlying hardware. The numbers given are valid for the GeForce 600 series from NVIDIA. Older GPU support only a maximum of 512 threads, where 16 threads are executed in parallel per block.



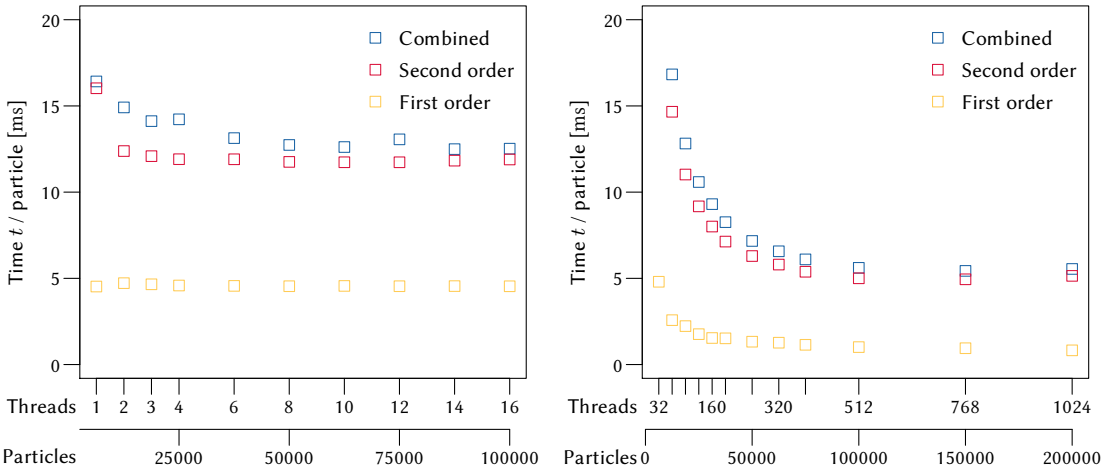
▶ **Figure 4.5 —** The closeup of the strong scaling results obtained with CUDA (Figure 4.4, right) reveals two local minima for 192 and 512 threads. The optima are visible for both, the total time including both reactions (blue, top) and the total time of second-order only reactions (red, bottom).

individual simulation parts are shifted, where 11.6 % and 80.7 % of the time are consumed for first-order and second-order reactions, respectively. The reduction in time for the random walk might be explained by the fact, that it is easily parallelizable and there are no dependencies. The implementation of first-order reactions on the GPU is slightly more complicated than on the CPU because newly spawned particles cannot easily be created. Hence, an expensive sorting and updating step is necessary. In total, the GPU implementation is about 1.8 times faster than the CPU implementation in this scenario.

Interestingly, more effort seems to be required on the CPU for first-order reactions, when they are performed while second-order reactions are also enabled. This is contrary to the expectations since both reaction types are independent and executed subsequently (cf. Section 4.2). This effect is possibly caused by some caching issues on the CPU. When only reactions of first order are performed, the complete data might fit into the CPU cache. In combination with second-order reactions, the memory footprint of the simulation grows due to the acceleration grid for the neighborhood search and might no longer fit into the cache, thereby leading to a degradation in performance.

On the GPU, a slight increase in performance is visible in the beginning when the number of threads is increased from the initial number of 32 threads. The measurements of the time spent for reactions is equal in all three runs as in contrast to the CPU version. This behavior is also to be expected because the number of particles is constant and the computational effort is the same, being completely independent from subsequent computations. In Figure 4.5, a closeup of the GPU measurements is shown. Two local minima are distinguishable for 192 and 512 threads that might be caused by some internal mechanisms of CUDA or the GPU. Considering the standard deviation of less than 0.1 ms per time step, the total time may vary by 6 seconds. For this reason, the existence of the two optimal points is less important when nearby data points of other threads are considered.

In conclusion, it can be said that both implementations, OpenMP as well as CUDA, do not exhibit strong scaling with the number of threads. However, there remain two open questions. First, the effect that independent reactions require more time when performed together on an OpenMP implementation and, secondly, the seemingly optimal points in the GPU simulation.



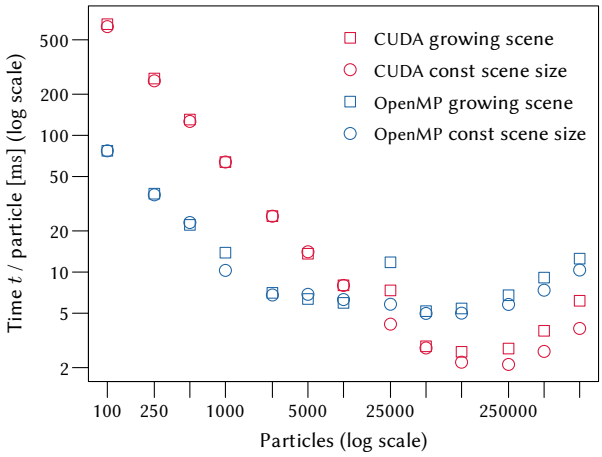
▲ **Figure 4.6** — Weak scaling performance test: OpenMP (left) and CUDA (right). The number of particles correlates linearly with the number of threads. CUDA measurements for 32 threads are not shown for combined (29.6 ms) and second order (25.4 ms) values. Parameters are given in Appendix A.3.

Weak Scaling

The model setup to show weak scalability is identical to the setup for strong scaling described in the previous section. Again, the thread count for OpenMP varies between 1 and 16, whereas for the GPU simulation the number of threads is set to multiples of 32. The major difference to the previous configuration is that the number of particles directly depends on the thread count. The baseline with 50 000 particles was chosen for 8 CPU threads and 256 GPU threads, respectively. The particle number is set to be proportional to the number of threads to test for weak, linear scalability, i.e. the average workload per thread is to be assumed constant. Hence, 100 000 particles are used in the simulation with 16, respective 512, threads. Since the molecule concentrations vary, the reaction rate constants have to be adjusted to maintain a constant rate of reactions per simulated time step (see Appendix A.3).

The normalized results, plotted in Figure 4.6, show the time spent per particle against the thread count and the particle number, since it is proportional to the thread count. The CUDA implementation benefits from higher particle counts as the performance increases from 29.6 ms with 32 threads to 5.5 ms with 512 and 1024 threads. This is mainly due to the under-utilization of the GPU. First-order and second-order reactions benefit both from the increasing utilization. On the CPU, the performance increases as well. The net gain, however, is due to the second-order reactions. The first-order reactions exhibit the assumed linear scalability and do not contribute to the performance gain, since the time spent per particle remains constant over different thread counts.

Both implementations feature a sublinear scalability concerning the number of particles with the exception of first-order reactions on the CPU, which scale linearly. At low particle numbers (6250), the CPU is almost twice as fast as the GPU. For high numbers of particles,



▶ **Figure 4.7** — Influence of the number of particles. For many particles, i.e. more than 20 000, the CUDA simulation outperforms the CPU simulation with an optimal performance for 250 000 particles. Parameters are given in Appendix A.3.

i.e. more than 100 000, the speedup obtained with the GPU compared to the CPU is about 2.3 fold for this particular scenario.

4.4.2 Varying the Number of Particles

In the previous section, the weak scalability of the parallel implementations is analyzed. The overall scalability is, however, only of secondary importance because the limiting factors, like the maximum number of concurrent threads, is given by the hardware that is used. For actual simulations, it is of more importance to know how the algorithm scales depending on the number of modeled entities like, e.g., the molecules contained in the model or the number of reactions. The following performance test hence deals with the number of particles in two similar, but distinct, scenarios.

The size of the simulation domain is constant in the first scenario, i.e. plasma membranes' diameter does not change. Since the cell volume remains constant irrespective of the particle number, molecular concentrations change proportionally to the number of particles. To counterbalance the higher concentrations, the reaction rate constants are adjusted accordingly to maintain a constant number of reactions per simulation step. In the second scenario, the reaction rate constants stay unchanged, but the radii of both the plasma membrane and the nucleus are determined to maintain a constant molecular concentration. Additionally, the number of cytoskeletal filaments is increased so that the occupied volume percentage of the cytoplasm remains constant at 1.5 %. The basic configuration of both scenarios is described in Appendix A.3. Both configurations are designed to be identical for 1947 molecules per species and with reactions rate constants $k_{ij} = 1 \times 10^7\,\mathrm{L\,mol^{-1}\,s^{-1}}$ for second-order reactions and $k_i = 1\,\mathrm{s^{-1}}$ for first-order reactions.

For the measurement itself, the number of threads was set to 8 for OpenMP and to 256 for CUDA. In total, 20 000 steps were simulated for particle numbers between 100 and 1 000 000. The results are shown in Figure 4.7. The break-even between CPU and GPU simulation is at about 20 000 particles where the GPU becomes faster. Until then, there seems to be a constant overhead, caused by high fix costs of GPU sorting during the

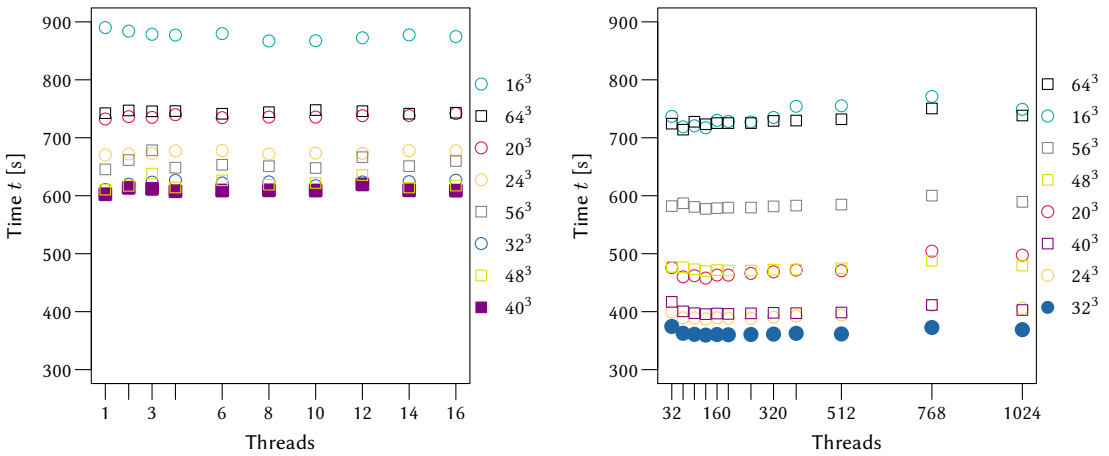
reactions, predominating all other computations. For high particle numbers these fix costs amount for less and the overall performance increases up to 250 000 particles. At this point the optimal performance is obtained with the current CUDA implementation reaching a speedup of 2.7 compared to OpenMP. The optimal point of the OpenMP simulation is already reached with 100 000 particles. Here, the optimum is, however, not as distinct as on the GPU.

The difference between the two scenarios becomes only evident for large particle numbers. The first scenario, i.e. the scenario with the constant scene size, performs better than the second one, where the plasma membrane grows in volume. This might be explained by the thread workload. Assuming a constant scene size, the number of particles per cell of the acceleration grid, which is used for the second-order reactions, increases with the total number of particles. Given a grid resolution of 30 in each dimension, all particles are distributed into the grid cells enclosed by the spherical plasma membrane. Inserting each particle only once results in an average occupancy of 0.74 particle per cell for a total of 10 000 particles. Even if each particle is contained in eight cells at once, the average workload per thread is pretty low. If the particle count is now increased, the grid cells will contain more particles and, therefore, more potential reaction partners. Thus, the workload per thread is increased and the utilization is improved.

4.4.3 Dimension of the Uniform Grid

The computation of second-order reactions relies on an efficient neighborhood search for potential reaction partners. In the current implementation, the neighborhood search uses a uniform grid as acceleration structure. The same grid is also used to parallelize the second-order reactions in a locally independent way. The dimensions of this grid, therefore, play an important role influencing both, parallelization and neighborhood search. A high grid resolution reduces the workload per thread, because of fewer particles per cell due to the fine subdivision of the simulation domain. On the other hand, if the resolution of the grid is too low, there might be less grid cells to compute than threads are available. Therefore, the effect of different grid resolutions is measured while varying the thread count as well. The simulation configuration is a cell with 50 000 particles including a cytoskeleton (cf. Appendix A.3). For each simulation run, 50 000 steps were performed to measure the total time. The distinct results are depicted in Figure 4.8 for OpenMP (left) and CUDA (right). Both measurements exhibit a unique, optimal grid resolution. The number of active threads plays only a minor role compared to the number of grid cells. The total time varies depending on the thread count, but the overall effect is almost negligible except for the first three measurements with OpenMP. The optimal grid resolution, however, is in both cases not affected by the thread count.

Choosing the correct grid resolution is of great importance to achieve the optimal performance as can be seen with this example. In this test scenario, the performance can be improved with the optimal grid dimension by a factor of 1.5 for the CPU and a factor of 2 for the GPU when compared to the respective worst cases.



▲ **Figure 4.8 —** Influence of the acceleration grid size: OpenMP (left) and CUDA (right). The optimal grid size is highlighted by filled markers. Parameters are given in Appendix A.3.

4.5 Signal Transduction Pathways

For the simulation only raw performance numbers have been discussed so far. These numbers might be of interest to estimate the total runtime of the simulation but they are only of minor significance to biology experts, who are more interested in the outcome of the simulation. Therefore, four scenarios were designed to qualitatively demonstrate the applicability of the simulation to biological questions and different aspects of the simulation.

The first example is a simplified MAPK pathway with only one mobile activated protein stage to study the influence of the protein transport mechanisms. The active proteins are transported either by diffusion or using the motorized transport along the filaments of the cytoskeleton. The second scenario covers all three major tiers of the generalized MAP3K pathway (cf. Section 2.1.2). Here, the number of particles is much larger and there are twelve reactions instead of three when compared to the simplified MAPK pathway. The two remaining scenarios demonstrate the import of proteins into the nucleus by nuclear pore complexes and the effects of protein activation on a dynamic cytoskeleton. In the following, the different scenarios are explained in detail and the performance results of the simulation runs are given.[4]

4.5.1 Simplified MAPK Pathway

The MAPK pathway usually consists of three tiers as described earlier in Section 2.1.2. The upstream part of the signaling cascade, however, is assumed to be located near the plasma membrane. In this model, the upper two tiers, i.e. MAP2K and MAP3K, are, therefore,

[4] Time measurements in the text are either given in seconds or in the format $i:j:k$ h representing i hours, j minutes, and k seconds.

omitted and are replaced with a single receptor complex yielding a simplified MAPK pathway model as described by Falk et al. [2009].

Instead of being activated by MAP2Kpp, the MAPK is directly activated by a membrane-bound receptor complex. The activated form of MAPK, i.e. MAPKp, either diffuses through the cytoplasm toward the nucleus or is actively transported along filaments of the cytoskeleton by motor proteins. The regulation of the signal includes the opponent deactivation of MAPKp by phosphatases. Thus, the signal translocation to the nucleus depends on the arrival of MAPKp, despite the dephosphorylating reaction everywhere in the cell. The local excitation at the plasma membrane together with the global inhibition leads to spatial gradients; only a few MAPKp molecules will reach the nucleus. If motor proteins transport the MAPKp along the cytoskeleton directly to the nucleus, this will change the MAPK distribution and more molecules will arrive at the nucleus [Kholodenko, 2003].

The simulation, containing 70 000 particles in total, was performed with Δt set to 9×10^{-7} s for 2 million steps yielding a simulated time of 1.8 s. The model setup including the distinct molecule counts and the reaction rate constants is listed in Appendix A.4.

Results

Using the OpenMP implementation, one simulation run took 9:10:18 h on the CPU. The GPU version using CUDA was almost twice as fast requiring only 4:38:09 h. Using both transport mechanisms, diffusion and transport by motor proteins, about 1500 molecules of MAPKp arrive at the nucleus and adsorb themselves.

When the motorized transport is disabled and the adsorption of MAPKp into the nucleus is the only remaining interaction, the total time amounts to 4:33:23 h for CUDA. This is equivalent to a saving of 1.7 % per step due to the reduced number of possible interactions with cellular structures. In the end, only 25 molecules actually arrive at the nucleus.

4.5.2 Generalized MAP3K Pathway

In this pathway, the MAPK pathway including all three tiers is simulated. The model setup is based on the model of Markevich et al. [2006] (see Appendix A.5 for the detailed parameter set). The Michaelis-Menten kinetics of the model by Markevich et al. is translated into the event-based framework as described in Section 3.1.6. The nonlinear feedback loop in the original model contained the squared concentration of MAPKpp. Therefore a dummy species F is created in a feedback reaction with 2 MAPKpp $\longrightarrow$ 2 MAPKpp + F. These dummy molecules decay quickly so that the concentration of F is proportional to the squared concentration of MAPKpp.

The model of Markevich et al. [2006] contains extreme parameters. Especially the tiny K_m values result in huge bi-molecular reaction rate constants in the present framework. They actually exceed the diffusion-limit $\gamma_{ij} = 4\pi(r_i + r_j)(D_i + D_j)$, if the natural molecular radius of MAPK molecules and phosphatases and the given diffusion coefficient

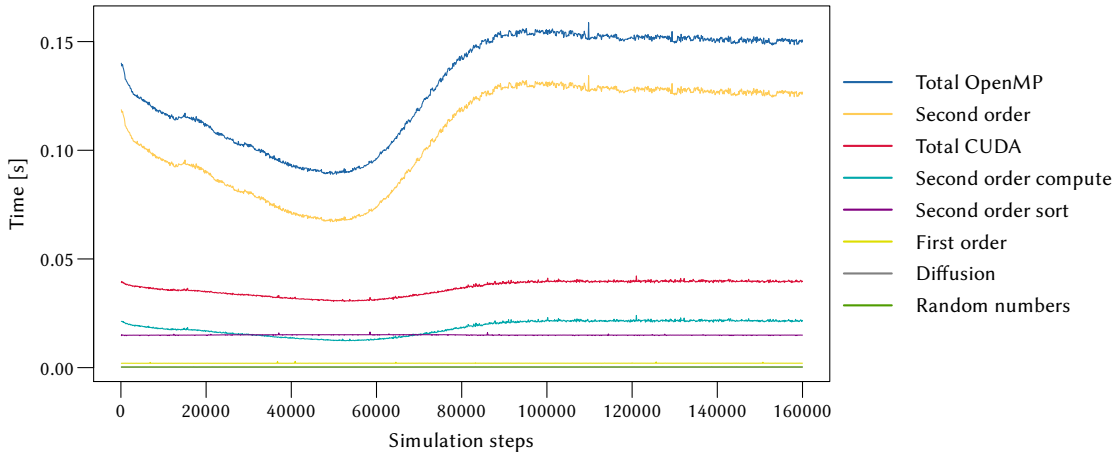
of $2 \times 10^{-12}\,\mathrm{m}^2/\mathrm{s}$ is used. In order to have a sufficiently high diffusion limit γ_{ij}, the particle radius is set to 20 nm in the present simulation. Another way to reduce the rate constant would be to increase the enzyme concentration ($k = v_{\max}/c_{\mathrm{E}}c$). The resulting large enzyme concentrations c_{E} would, however, be in contradiction with the quasi-steady-state assumption underlying the Michaelis-Menten kinetics. This fact shows, that the interesting phenomenon of traveling waves cannot be established that easily in nature as in the theoretical model of Markevich et al. [2006], because it requires quite extreme parameter values. The use of Michaelis-Menten kinetics for such models remains questionable with respect to the limitations of the underlying quasi-steady-state assumption [Segel and Slemrod, 1989].

The enzymatic deactivation of the MAP2K* molecules was replaced by a first-order decay to simplify the simulation. MAP3K is set to be always active while the upstream part of the cascade is omitted in the present model. The concentration of MAP3Kp corresponds to the steady state level of the original model [Markevich et al., 2006].

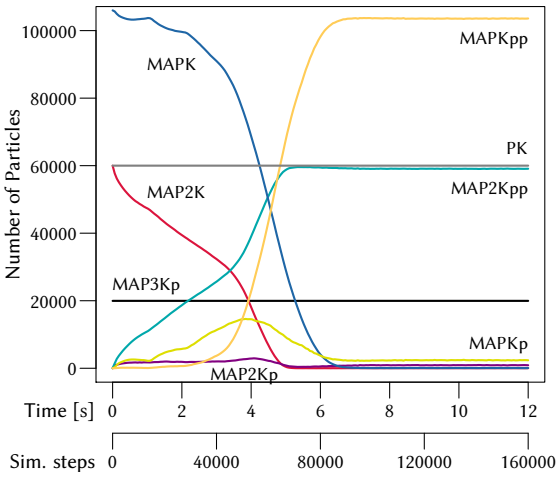
Results

The simulation was performed for a time interval of 12 s by simulating 160 000 steps. Using 8 threads, 5:46:06 h were required for the whole simulation on the CPU employing OpenMP. The implementation with CUDA was about 3.5 times faster, requiring only 1:37:29 h. The times expanded per simulation step are plotted in Figure 4.9 for both, OpenMP and CUDA. Detailed measurements are shown for the simulation run on the GPU. The generation of random numbers, the diffusion step, and the first-order reactions require little, constant time per iteration of the simulation. The drop in the curve of second-order computations can be explained by examining the molecular population. Figure 4.10 depicts the populations of various molecular species over time. After the passage of four seconds, the population of MAPKp molecules reaches its peak before declining steadily. This is due to the reaction delay τ, which is set to four seconds for phosphatases deactivating MAPKp molecules. From this point onward, the first phosphatases become again active and can, therefore, deactivate MAPKp. The activation of MAPKpp molecules, however, is strong enough at this juncture, so that the phosphatases are now also delayed by the deactivation of MAPKpp, but this time the delay is much shorter. Since both phosphorylated states of the MAPK molecule are deactivating the same phosphatase in a cooperative manner, their survival probability is increasing. This observation supports the findings of Mather et al. [2010] in coupled enzymatic processes. The dips visible in the curves of MAPK and MAP2K are also caused by the reaction delays of phosphatases, which are initially all in their active state.

A closer look at the ratios of the computational loads on the GPU during a single simulation step confirms that the calculation of second-order reactions dominates the computation by far (see Figure 4.11). About 40 % of the time spent on second-order reactions is required for updating the acceleration structure, which requires sorting. This sorting step only depends on the number of particles and the time needed per simulation step is, hence, constant (cf. Figure 4.9).

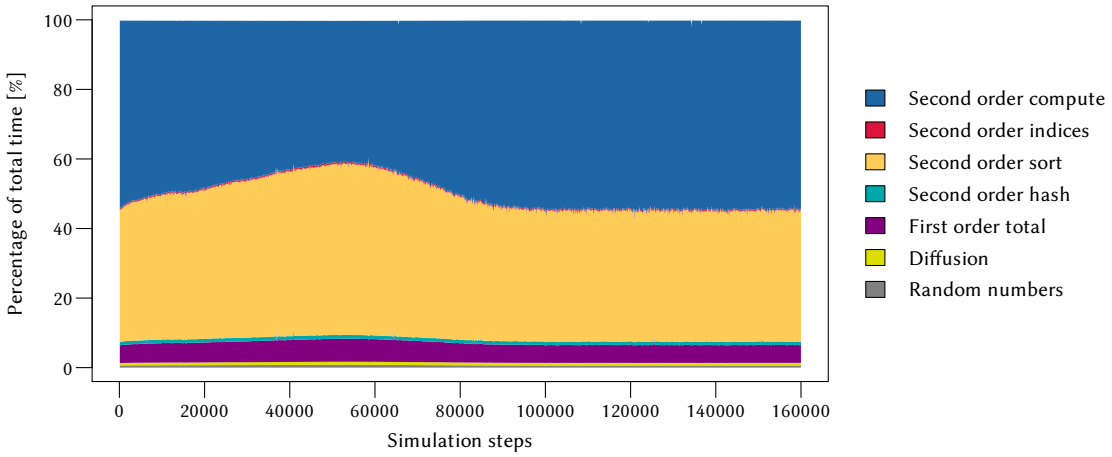


▲ **Figure 4.9** — Performance of the generalized MAP3K pathway. The time needed to compute second-order reactions is given for both, OpenMP and CUDA. The measurements of the remaining steps of one simulation pass are only shown for CUDA.



◀ **Figure 4.10** — The time series data for the generalized MAP3K pathway shows the development of the particle numbers of the different molecular species over the course of the simulation.

If two similar MAP3K simulations are performed on the same hardware at the same time, the results detailed above do no longer look so promising for the GPU. Previously, the OpenMP simulation required about 5:45 h. Now with two simulations running on the same CPU, they required 6:48:41 h and 6:48:51 h. This corresponds to a slow-down factor of 1.18, while performing two simulations at once. A control simulation with 16 threads instead of the original 8 threads resulted in 5:41:36 h. This behavior indicates, that large parts of the simulation do not fully utilize the CPU because of sequential code or race conditions. When performing the same experiment with one GPU, the outcome is almost as expected. Since one simulation already fully utilizes the GPU, the two simulations have to share the resources and the total time will go up. With two simulations simultaneously executing on the GPU, both simulations took 3:11:38 h yielding a slow-down of 1.97. If the two simulations executed in parallel, the original speedup factor of 3.5 is reduced to a mere two-fold speedup of the GPU over the CPU



▲ **Figure 4.11 —** Performance ratios of the MAP3K simulation on CUDA. The calculation of second-order reactions dominates the overall computation in every step of the simulation.

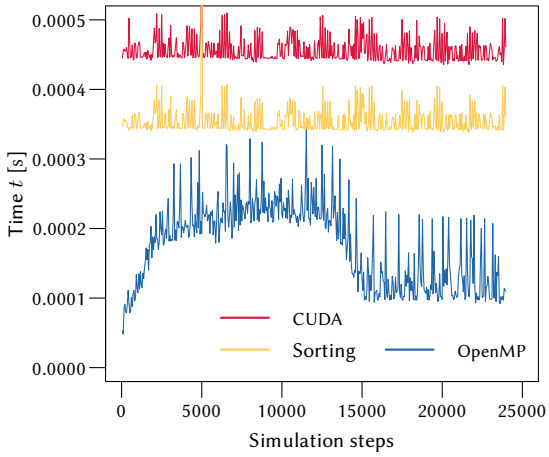
4.5.3 Protein Import by Nuclear Pore Complexes

This transport demonstrates the import and export of molecules through nuclear pores. The main difference to the motorized transport is that the surrounding compartment of imported molecules changes from cytoplasm to the inside of the nuclear envelope and vice versa for exported molecules.

Initially, two species of molecules exist. The molecular species C is located in the cytoplasm which surrounds the nucleus. The second species, i.e. N, is constrained to the inside of the nuclear envelope. Instead of over 2000 nuclear pores existing on the surface of the nucleus, only 500 pores are used and randomly distributed on the nuclear envelope. The binding rates for both species are set to rather high values of $k_C = 1\,000\,000\,s^{-1}$ and $k_N = 800\,000\,s^{-1}$, so that the overall effect of import and export becomes visible. The model was simulated for 700 000 steps resulting in about half a second of simulated time (see Appendix A.6 for model parameters).

Results

Of the 20 000 molecules of species C diffusing through the cytoplasm, about 350 got imported into the nucleus by the nuclear pores and almost all of type N, i.e. 170 molecules, got exported from the nucleus. Despite the low count of 20 200 molecules in the scene, the GPU outperforms the CPU by a factor of 10.67. On the GPU only 83 s were necessary to complete the simulation, whereas the CPU simulation required 14:45 min. The rather high speedup is mainly due to the fact, that only particle diffusion and structure interactions are included in the model. Since the particle number does not change throughout the simulation, the total time needed per iteration is constant with 1.25 ms for OpenMP and 0.12 ms for CUDA.



◀ **Figure 4.12 —** Timings of the dynamic skeleton update step. The time spent on the calculations of the membrane concentration and the skeleton update is constant for CUDA (red), whereas the computations are dependent on the number of active particles in OpenMP (blue). The sort step (yellow) is part of computations in CUDA.

4.5.4 Dynamic Cytoskeletal Filaments

To model a dynamically adjusting cytoskeleton, an external, extracellular stimulus is required. This external stimulus is modeled by a single pulse sweeping over the plasma membrane of the cells, where it changes the activation state of membrane-bound receptors. Initially, all receptors are in their inactive state and the lengths of the cytoskeletal filaments are set to zero. During the simulation, the local concentrations of activated receptors are used to influence the growth and shrinkage of individual filaments of the cytoskeleton (cf. Section 3.1.9). The cytoskeleton itself consists of microtubules, which are placed radially at the nucleus. The length of the microtubules is adjusted dynamically during the simulation. The detailed model configuration can be found in Appendix A.7.

Results

The simulation required 174 seconds on the CPU and 159 seconds on the GPU resulting in a speedup of 1.10 from CPU to GPU. The main bottleneck of the simulation is the update of the dynamic skeleton, which incorporates the estimation of molecule concentrations on the plasma membrane. In Figure 4.12, the total time needed for the update step is compared between OpenMP and CUDA. The most obvious conclusion is that the GPU-based implementation requires a constant time throughout the whole simulation while on the CPU the load varies. In fact, the variation in time on the CPU directly correlates with the number of activated particles since they are sequentially processed (cf. Section 4.2.8). On the GPU, all particles have to be processed and selected by their activation state. The sorting, which is a necessary part of the GPU implementation, accounts for about 75 % of the total costs. The evolution of the cytoskeleton over time is depicted in Figure 3.6.

CHAPTER 5

VISUALIZATION TECHNIQUES FOR SYSTEMS BIOLOGY

Life sciences depend on imaging and image analysis to a significant extent. Body scans with computed tomography and microscopy of individual cells provide information about the structure and morphology and some function. In contrast, biochemistry and genetics allow for a deep insight into the molecular machinery, their regulation, and eventually their function. Mathematical models integrate this data, test new hypothesis for biological functions, and can predict diseases. Nevertheless, this systems biology approach for model development can only lead to a better understanding of the biological processes, if it is possible to visualize the results and relate them to the images and outcomes of experiments. Visualization is, therefore, needed to have an intuitive understanding of the data resulting from the models.

In signal transduction processes like the MAPK pathway, biologists are traditionally only interested in the number of activated molecules at a given time. The behavior of the biological system is shown in protein concentration profiles over time and space. These profiles can include 3D graphs of the concentrations with respect to parameter changes [Kholodenko, 2003]. Two- or three-dimensional plots are employed to visualize the trajectories of single proteins [Howe, 2005; Hazelbauer, 2005; Lipkow et al., 2005]. Instead of graphs, new 2.5D and 3D visualization techniques are proposed in this thesis that allow for the interactive exploration and visual analysis of the environment of the virtual cell. Established visualization techniques like glyph-based methods and volume rendering are combined in new ways and are adjusted pertaining to the data resulting from cellular simulations. Since cell simulations include thousands of proteins, obstacles, and filaments, the visualizations tend to include visual clutter. Diffusing proteins are hence extremely difficult to track. This becomes worse by their chaotic motion. One aim is, therefore, to highlight the events of interest in the

confusing orchestra of structures and background molecules by different means. The human perception is guided by depth of field and other depth cues to the objects in focus.

Schematic cellular visualization [Falk et al., 2009] shows the data in a representation as it is used throughout the simulation process, i.e. spherical molecules and cylindrical microtubules and microfilaments. This visualization relies heavily on 3D glyphs since glyph-based rendering allows the interactive display of millions of objects. Atomistic visualization [Falk et al., 2012] builds upon the schematic visualization by refining the structures down to the atomic level of both proteins and cytoskeletal structures. Both visualization approaches represent the scene using underlying geometry, but that is not how cells are usually perceived in wet lab experiments. Studies on signal transducing proteins often rely on CLSM for producing images to locate and count salient proteins. The aim of the microscopic visualization lies, therefore, in mimicking such images from CLSM to allow for comparisons between *in silico* simulations and *in vivo* or *in vitro* experiments.

Since the data resulting from the simulation is particle-based, it is difficult to estimate local differences in the molecule concentrations, i.e. the respective distributions of particles. When switching from the discrete particle representations to a continuous representation, differences in molecule concentrations become visible [Falk et al., 2010b]. Here, a volumetric representation of the particle data is computed and then visualized with volume rendering techniques. The image-space approach, which is subsequently described, is an enhancement of that method. Instead of computing an intermediate volume representation, the volume data is reconstructed and displayed on-the-fly during the visualization of the particle data [Falk et al., 2010a]. Thereby, sampling artifacts are reduced and the memory footprint is kept low. The general applicability of both approaches is demonstrated in the respective sections since both are applicable to generic particle data and not only the cellular environment. The technique presented last is the visualization of membrane-bound receptor clustering [Falk et al., 2011a]. Here, the concepts of the schematic cellular visualization are adapted to the use case of detecting protein clusters on a small extent of the plasma membrane.

5.1 Getting Insights and Supporting Visual Perception

Especially when examining events deep inside the cell, a cluttered space hides the objects or obscures events of interest. To gain insights it is, hence, important to reduce visual clutter and enhance visual perception. The three interactive stages of the visualization pipeline are the key to obtain a visually pleasant and informative representation (cf. Section 2.5). To limit the overall crowding, unwanted molecules can be filtered out in the filtering stage of the pipeline either by logical or spatial segmentation. One of the logical criteria for segmentation is the particle type. Spatial segmentation is performed by applying cut planes during rendering, although this is considered to be part of the filtering stage. The radii of molecules might be re-estimated in the mapping stage and more elaborate techniques, like shading, depth cues, and ambient occlusion, can be applied in

the later stage of the actual rendering. A combination of these approaches is possible and sometimes advisable when visualizing the data.

5.1.1 Cuts and Transections

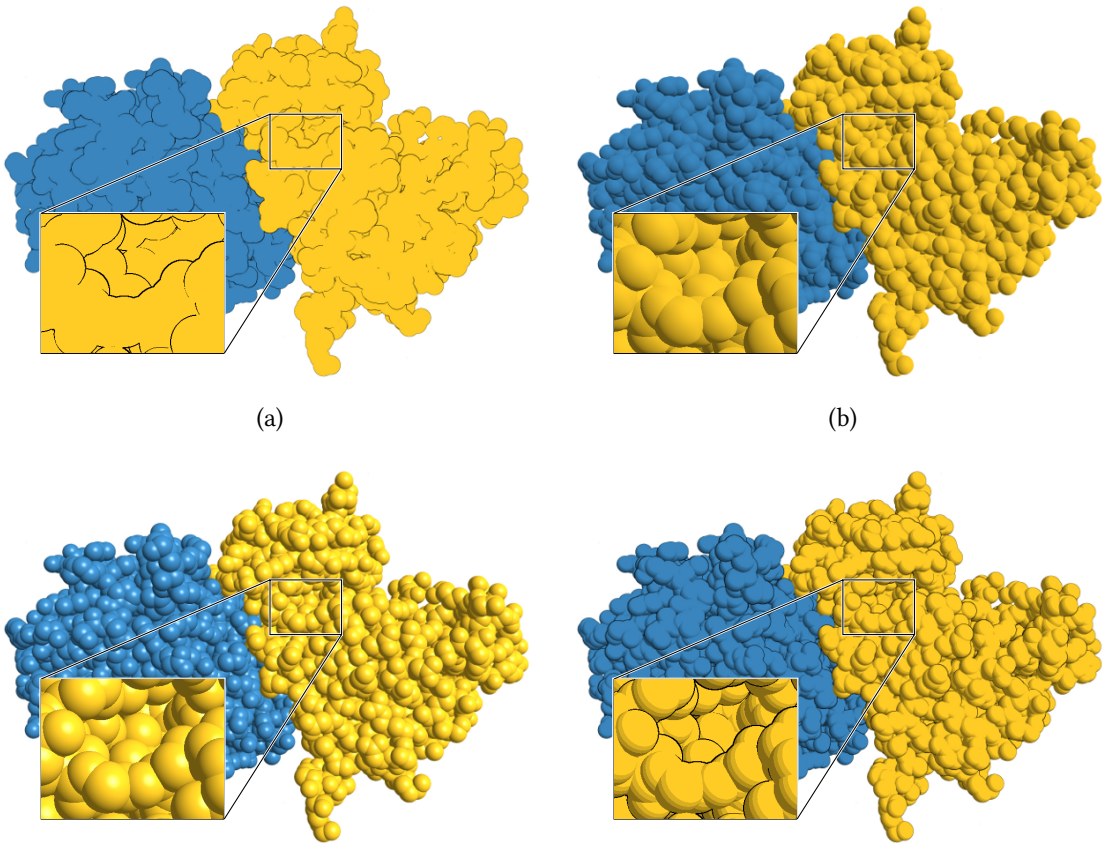
Although transparency allows one to look through rendered objects, it creates a haze around shapes that only disorients the eye. In this case, it is much more helpful to cut away the perturbing parts of the cell to allow a clear view at the object of interest. The cutting is realized with a cut plane, also called clipping plane, that divides the 3D space. The content on one side of the plane is preserved whereas everything on the other side is clipped (cf. Section 2.4.1). The overall cutting process takes part during the rendering and is fully supported by the GPU, thus eliminating the need of any special data handling.

To improve the flexibility of cut planes, each cut plane can be restricted to cut only a specific, user-defined set of scene objects. This set might include the filaments of the cytoskeleton or some specific particle types. By allowing the user to define the locations and properties of several cut planes, the general concept allows to remove occluding or disturbing parts. Thus, visual clutter is greatly reduced. A typical scenario for the application of cut planes are cellular transections, i.e. a thin slice. If two cut planes are aligned parallel to each other, a small transection of the cell can be realized. Since such transections possess only a small depth, they can provide a quick overview of the data without too much crowding and also allow to compare multiple time steps of time-varying data.

5.1.2 Shading

Shading drastically affects the perception of shapes and objects. Depending on the used technique, shading defines the appearance of objects where they might appear flat or spatially extended (see Figure 5.1). This property is of high relevance, as can be seen in the field of computer vision, where the object shape is inversely derived from the object appearance in an image [e.g. Rindfleisch, 1966]. Another important object property is encoded in the shading, the object material properties. Different shading models exist to model a multiplicity of isotropic and anisotropic materials resulting in matte, plastic-like, or velvet-like materials.

The simplest shading is flat shading, where a single constant color is assigned to the object resulting in an overall flat appearance. Effects of diffuse lighting, i.e. non-glossy objects, can be incorporated with the Lambertian model [Lambert, 1760]. Typically, only the angle between surface normal and the incident light is considered yielding a smooth object appearance. For rough, diffuse objects like the moon or matte materials, the model by Oren and Nayar [1994] is better suited since it takes the roughness of the material into account. The Phong [1975] model comprises a specular term for highlights besides the terms for diffuse and ambient shading. Instead of calculating the exact reflection, the specular reflection is approximated in the Blinn-Phong shading model by considering the

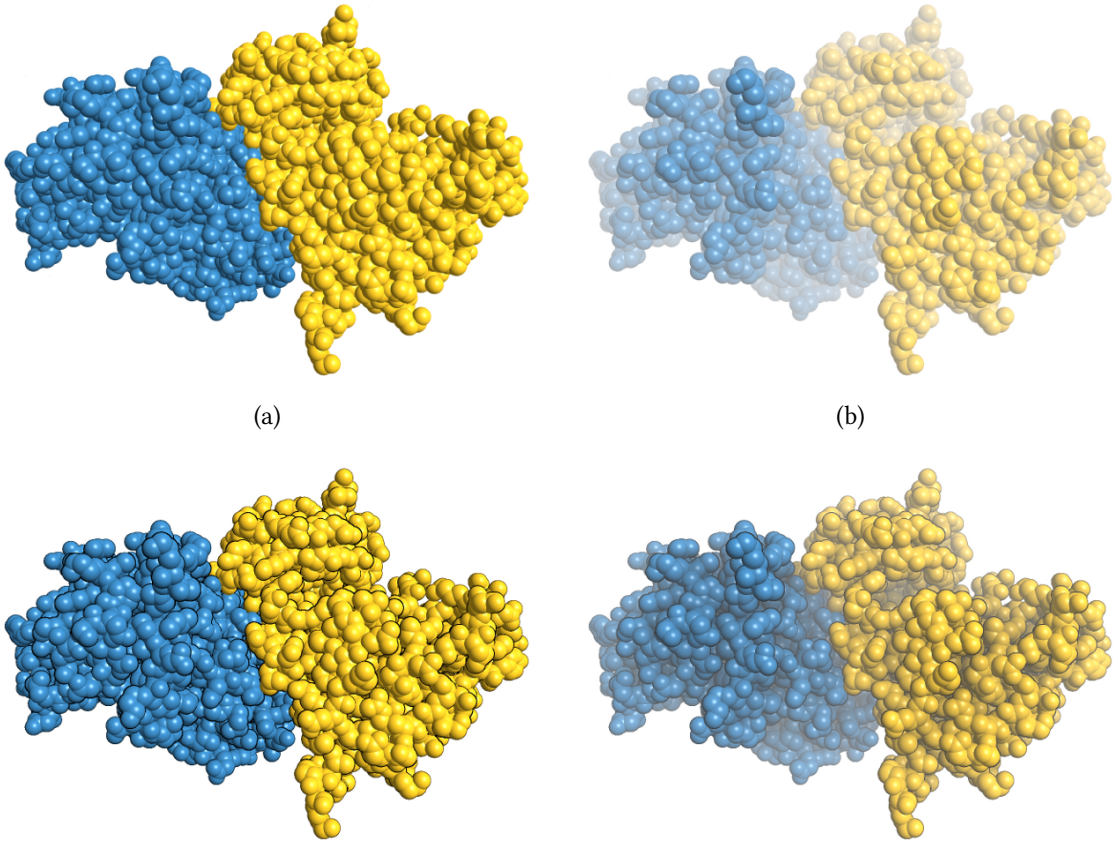


▲ **Figure 5.1 —** Comparison of shading techniques. (a) flat shading with silhouettes. (b) Oren-Nayar model. (c) Phong illumination. (d) toon shading.

viewing direction and the direction of the incident light [Blinn, 1977]. Toon shading can be used to produce artistic or non-photorealistic renderings. In contrast to the previously mentioned models, toon shading does not describe the material properties. The results of an arbitrary shading model can be used as input for toon shading. The colors are segmented into a couple of distinct shades to produce a comic-like look instead of smooth color gradients. In Figure 5.1, the different shading models are applied to the atomic structure of a protein (B-Raf kinase, PDB ID: 3PPJ).

5.1.3 Depth Cues, Halos, and Silhouettes

Cues aiding the visual system in perceiving differences in depth are called depth cues. The most prevalent depth cue is the object size. If the object size is known for a certain depth and the same object appears smaller in a subsequent time step, it must be further away. The same holds if similar objects appear in the same time step at different depths. With a narrow field of view or an orthographic camera, however, the size of an object no

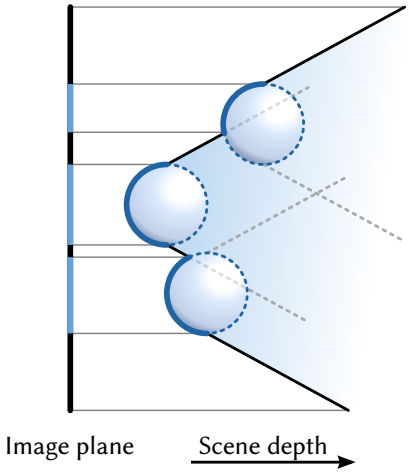


▲ **Figure 5.2** — Comparison of depth cues to support depth perception. (a) original scene without depth cues. (b) depth cues by desaturation. (c) silhouettes obtained by edge detection. (d) combined depth cues with object-space ambient occlusion (OSAO).

longer varies noticeably with increasing depth and additional depth cues are necessary. Depth cues from motion parallax, i.e. objects closer to the camera appear to move faster, are inherently useful for a sequence of images but are not applicable to static images.

Different depth cues applied to the same scene are shown in Figure 5.2. A depth cue experienced in nature is the atmospheric perspective, where distant objects appear more bluish and less saturated due to the atmosphere of the earth. This type of depth cue is considered as a color depth cue in the context of this thesis. Color depth cues assist the human perception by accentuating the colors of objects according to their depth. In this work, desaturation and a slight increase in brightness is used to emphasize depth differences and to blend distant objects with the background (Figure 5.2(b)). This is done by a linear transformation into 3D tristimulus color space XYZ and back in the respective fragment shaders as described by Weiskopf and Ertl [2002].

Besides color and size, distinct object boundaries are a beneficial depth cue for scenes with many similar objects, like the visualization of cellular simulations. Silhouettes



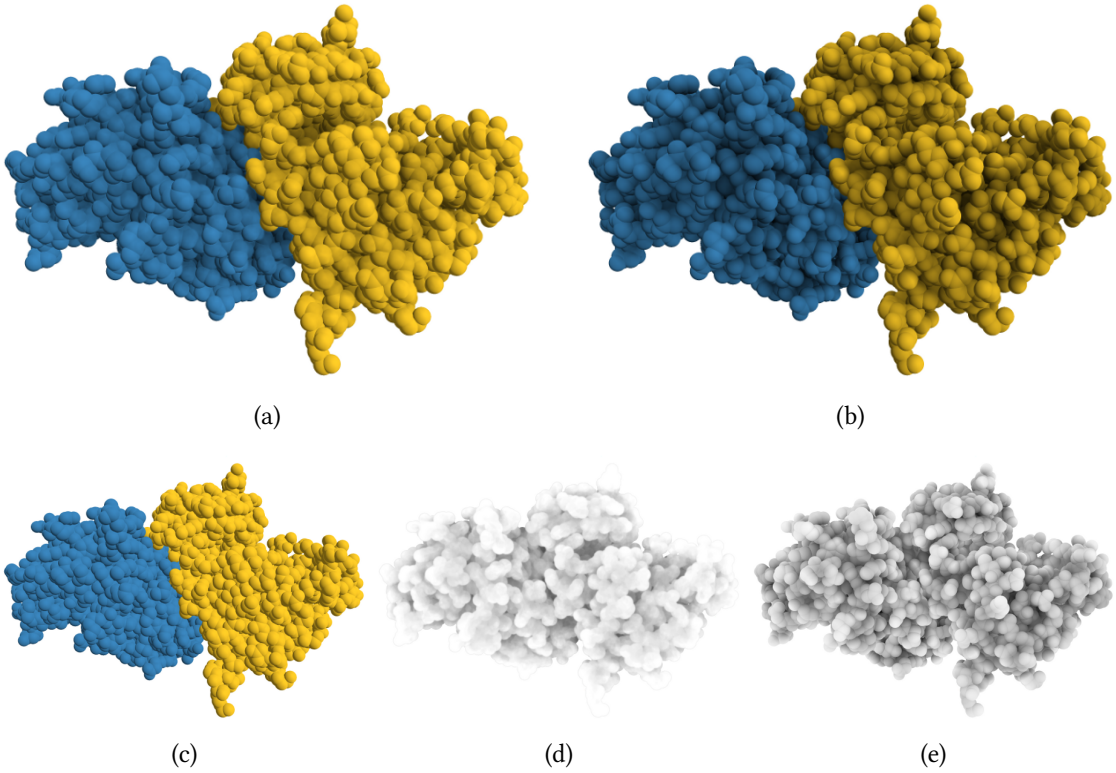
◀ **Figure 5.3 —** Depth-dependent halos applied to three particles. Only parts of the particles are visible due to the halos. The shaded area indicates the scene parts that are occluded by the particles and their halos. Illustration according to Tarini et al. [2006].

are a simple, yet effective cue to emphasize spatial differences and to separate objects. Depth-dependent silhouettes [Saito and Takahashi, 1990] are computed in image space in a postprocessing step. The depth values of the original image are processed with a Sobel operator to detect edges, i.e. discontinuities in depth. The line widths of the silhouettes are then adjusted depending on the differences in depth. The resulting silhouettes are thin or non-existent for objects at similar depths and thicker at the boundaries of objects that are far apart (Figure 5.2(c)). A similar effect is obtained by applying halos extending from the object boundaries [e.g. Tarini et al., 2006]. At the boundary of the object, the halo features the same depth as the object. With increasing distance from the object, the depth of the halo increases as well. In Figure 5.3, this approach is applied to three particles. The halo cones of nearby, spherical objects intersect with each other, resulting in a narrow halo whereas distant objects retain their complete halo. The depth perception can be additionally aided by combining the different depth cues (Figure 5.2(d)). Further depth cues including ambient occlusion to emphasize local depth differences and depth of field are described in the following sections.

5.1.4 Ambient Occlusion

Shading models presented earlier incorporate only direct illumination, i.e. effects like occlusion, shadowing, and color bleeding are not considered. Solving the rendering equation [Pharr and Humphreys, 2010] would include such effects but the computational effort needed is tremendous. A less sophisticated but fast way is to approximate these global illumination effects. One class of approximation approaches is concerned with ambient occlusion. Ambient occlusion methods mimic the transport of diffuse light between scene objects leading to localized shadowing in creases. The result is a non-physical effect in contrast to solving the rendering equation to obtain physically correct, but computationally expensive, results.

Screen-space ambient occlusion (SSAO) is an image-space technique to calculate the effects of ambient occlusion in a postprocessing step. An efficient GPU implementation for this



▲ **Figure 5.4** — Screen-space and object-space ambient occlusion. (a) screen-space ambient occlusion (SSAO). (b) object-space ambient occlusion (OSAO). (c) original scene without ambient occlusion. (d) ambient occlusion factors of SSAO. (e) ambient occlusion factors of OSAO.

technique is possible [e.g. Kajalin, 2009; Ritschel et al., 2009]. The SSAO approaches work solely on the contents of the depth buffer. For each fragment an ambient occlusion factor is determined for the local neighborhood. A hemisphere is placed around each fragment and sampled randomly. The ratio of occluded samples, i.e. samples which lie behind the respective neighbor fragments, to the total number of samples determines the ambient occlusion factor. The occlusion factor can then be used in the subsequent deferred shading passes to affect, i.e. darken, the colors. The artifacts due to the stochastic sampling can be remedied by filtering the resulting SSAO texture, e.g. with mipmapping, smoothing it with a filter kernel, or both. Depth darkening as proposed by Luft et al. [2006] or horizon-based ambient occlusion by Bavoil and Sainz [2009] are two different approaches to obtain SSAO.

Screen-space approaches tend to create imperfect and incorrect shadowing since they can consider only the visible, or front most, parts of the scene. This fact can be remedied by employing object-space methods, i.e. object-space ambient occlusion (OSAO). In Figure 5.4, SSAO and OSAO are applied to the same scene and compared with the original scene containing direct illumination only. The calculated ambient occlusion factors are depicted in Figure 5.4(d) and (e), respectively.

(a) (b)

◀ **Figure 5.5** — Examples of the utilization of depth of field (DoF) in photography and computer graphics. (a) macro photography of flowers. (b) computer-generated scene with DoF.

The OSAO approach was first introduced by Zhukov et al. [1998] and was recently adapted by Grottel et al. [2012] to apply ambient occlusion effects onto dynamic particle data sets at interactive frame rates. Grottel et al. use a volumetric approximation to store the ambient occlusion factors. To compute these factors, a uniform grid enclosing the complete scene is used where each grid cell denotes the number of particles it contains. The uniform grid is stored in a 3D texture and is accessed during glyph rendering. The texture fetch takes place before the actual illumination calculation of the glyph by offsetting the current intersection point along the analytical glyph normal.

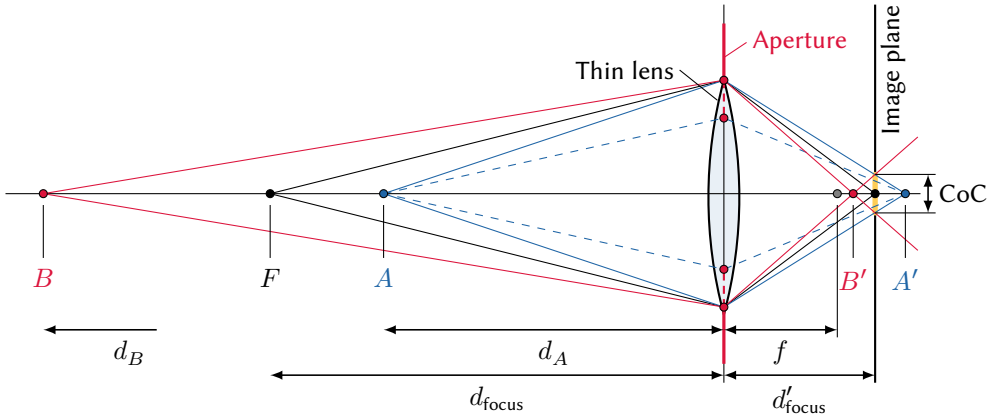
Both ambient occlusion approaches, in screen space and object space, have in common that they work only for very dense particle data sets. For sparse data sets, the effect of SSAO is barely visible whereas OSAO leads to a uniformly distributed occlusion factor or no occlusion depending on the parameters.

5.1.5 Depth of Field

Depth of field (DoF), as known from photography, can be used to separate foreground and background. The object of interest is focused whereas the remaining parts of the scene appear more or less blurred depending on the used camera settings. Typical examples include portrait photography or macro photography as depicted in Figure 5.5.

Most interactive DoF techniques in computer graphics rely on postprocessing techniques and reuse information from the color and depth buffers, an approach first proposed by Potmesil and Chakravarty [1981]. For a DoF approximation, heat diffusion [Kass et al., 2006] or a simple blur [Hammon, 2007] can be used. In visualizations, DoF can be used as focus and context technique to draw the observer's attention to certain aspects of the data set. The concepts of DoF can be extended to more general, non-physical methods used in visualizations, like, e.g., a semantic depth of field [Kosara et al., 2001]. Here, regions of interest remain in focus independent of their spatial location.

Potmesil and Chakravarty [1981] suggest a single-lens camera model to simulate the phenomena of DoF. Under the assumption of a thin lens, the image distance of an object



▲ **Figure 5.6 —** Thin lens model. The circle of confusion (CoC, orange) of a point on the image plane depends on both, object distance and aperture. A smaller aperture leads to a smaller CoC (dashed lines).

can be obtained with the thin lens equation

$$\frac{1}{f} = \frac{1}{d} + \frac{1}{d'}, \tag{5.1}$$

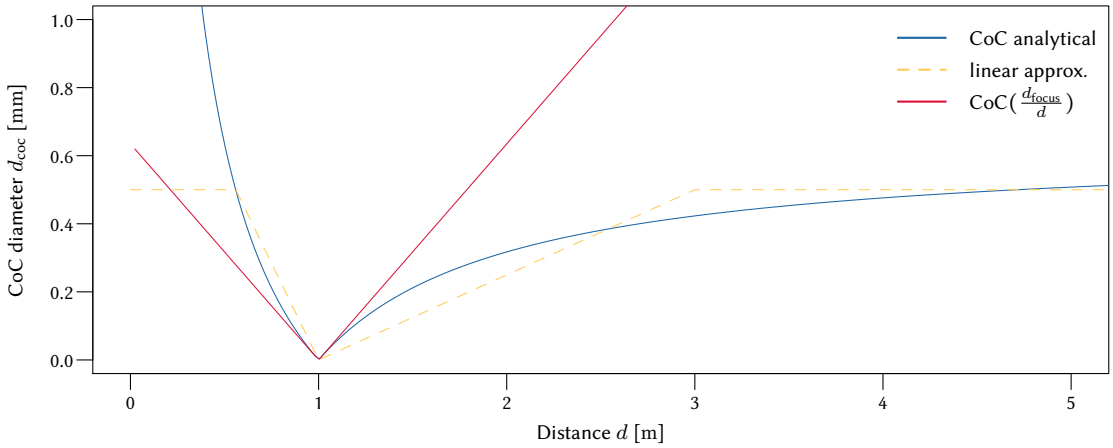
where f is the focal length of the lens, d describes the object distance to the lens, and d' is the distance of the image behind the lens. The focal length of the lens defines the point where parallel incident light is focused. Assuming a point F at distance d_{focus} to be in focus, its image has to lie directly in the image plane. Using (5.1), the location of the image plane at d'_{focus} is obtained by

$$d'_{\text{focus}} = \frac{f\, d_{\text{focus}}}{d_{\text{focus}} - f}. \tag{5.2}$$

In Figure 5.6, the thin-lens model is depicted with F being in focus. Since the points A and B do not lie in focus, their images lie in front and behind the image plane, i.e. B' and A', respectively. The result is a blurred circle of either point on the image plane, which is called circle of confusion (CoC). The diameter of the circle is dependent on the object distance and the used aperture. The aperture determines the amount of light passing through the lens by covering parts of the lens. In photography, the lens aperture is often expressed as f-number a instead of the aperture diameter d_{a}. There is, however, a direct relation between both, namely $a = \frac{f}{d_{\text{a}}}$. The aperture directly affects the size of the CoC of unfocused points as illustrated in Figure 5.6. The diameter of the CoC d_{coc} is calculated for a point at distance d by [Potmesil and Chakravarty, 1981]

$$d_{\text{coc}}(d) = \left| \frac{f\, d}{d - f} - \frac{f\, d_{\text{focus}}}{d_{\text{focus}} - f} \right| \left(\frac{d - f}{a\, d} \right). \tag{5.3}$$

From this equation clearly follows that the CoC for a given distance d is only dependent on the focal length, the aperture, and, of course, the focus distance. Figure 5.7 shows the



▲ **Figure 5.7 —** Diameter of the circle of confusion (CoC). When the CoC is calculated for the ratio of d_{focus} to d, the CoC is proportional to this ratio (red). The piecewise linear approximation of the CoC (orange) is used in the approach by Scheuermann and Tatarchuk [2004]. Parameters: $f = 0.035\,\text{m}$, $a = 2$, and $d_{\text{focus}} = 1\,\text{m}$.

diameter of the CoC for a given parameter set. At the focal distance d_{focus} of 1 m, the CoC function is discontinuous and evaluates to zero, i.e. yielding a precise image on the image plane. For smaller values the diameter increases drastically whereas for larger distances the CoC diameter converges. In the approach of Scheuermann and Tatarchuk [2004], the CoC diameter is approximated with a piecewise linear function into four segments to prevent the evaluation of Equation 5.3. Throughout the work presented in this thesis, however, the precise diameter is used. When considering the CoC diameter of the ratio d_{focus} to d instead of simply using the distance d, a linear relation emerges (see red line in Figure 5.7). More interestingly, the slopes of both lines are equal, apart from the sign. Using (5.3) and let d be $\frac{d_{\text{focus}}}{2}$, the slope m is given by

$$m = \left| \frac{f}{1 - 2\frac{f}{d_{\text{focus}}}} - \frac{f\,d_{\text{focus}}}{d_{\text{focus}} - f} \right| \left(\frac{1 - 2\frac{f}{d_{\text{focus}}}}{a} \right), \tag{5.4}$$

where the slope is independent of the distance d. With Equation 5.4, the CoC diameter can now be expressed with

$$d_{\text{coc}}(d) = \left| \frac{d_{\text{focus}}}{d} - 1 \right| m. \tag{5.5}$$

Thus, the CoC calculation is reduced to a mere subtraction, one division, and one multiplication with the precalculated slope.

Technique

The DoF approach is realized by an image-based postprocessing technique. The original GPU-based algorithm proposed by Scheuermann and Tatarchuk [2004] in the ShaderX

series was extended with respect to the exact calculation of the CoC diameter according to Equation 5.5. In addition, the proposed implementation employs a level-of-detail scheme, whereas the original method uses only one downscaled image.

The general idea is as follows. The scene is rendered without any constraints and the results of the color buffer and the depth buffer are both stored in textures. A mipmap chain[1] is created from the original contents of the color buffer by using bilinear interpolation similar to the method of Lee et al. [2009]. To reduce the artifacts of the interpolation, the result can be smoothed with a Gaussian filter kernel.

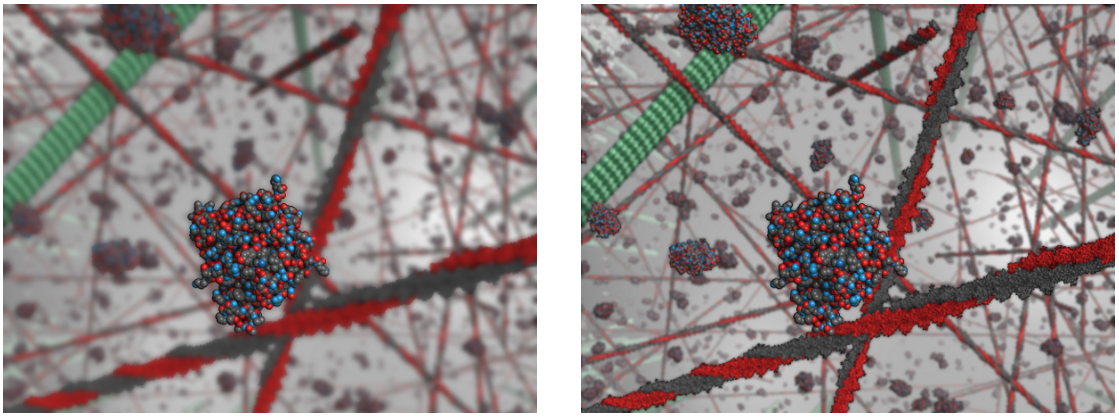
The DoF effect is then computed in a separate fragment shader. Using Equation 5.5, the CoC diameter is calculated depending on the focal distance d_{focal} and the respective depth d of the current fragment. The diameter is used to define a circle around the fragment limiting the area of contributing neighbors. To compute the contribution of the neighbors, the circle is sampled with a predefined number of sample positions, which originate from a precomputed Halton [1964] sequence or a Poisson [1837] distribution. For every sample the respective mipmap level of the color texture is fetched and accumulated. The level at which the mipmap color texture is accessed is determined also from the CoC diameter. Lower weights are used to reduce color leaking from samples that are closer and more in focus than the central fragment.

Results

Figure 5.8 illustrates the effect of DoF. The opening of the aperture affects the strength of the effect as expected (Figure 5.8(a), (b)). A larger f-number, i.e. a smaller opening diameter, enhances the size of the focal field. By choosing a distinct focal point, like on the microfilament in Figure 5.8(c), the attention is automatically drawn to the microfilament while the intracellular context is provided but only in a moderate manner.

The time needed for the DoF approach was measured for a complex scene (Figure 5.8(a)) and a simplified scene without the details on the atomic scale. Table 5.1 compares the results for the original method by Scheuermann and Tatarchuk [2004] and the approach described above, which employs mipmaps. At a resolution of 1600×1200, the DoF effect consumes 1.37 ms independent of the scene. This is not surprising, since the approach is carried out as a postprocessing step reusing only information from image space. The time spent on filtering the mipmap levels with a Gaussian kernel is constant, since it only depends on the viewport resolution and hence the number of mipmap levels. Also 0.24 ms are necessary for the filtering. The DoF approach by Scheuermann and Tatarchuk [2004] seems to suffer from a performance penalty when more than 12 samples inside the CoC are fetched. This adverse effect and the overall worse performance compared to the proposed novel approach might arise from the limitation imposed by the maximum texture memory bandwidth of the GPU. Scheuermann and Tatarchuk [2004] use only two textures, the original image and a downscaled version. Therefore, a larger memory

[1] From Latin *multum in parvo*: The original texture is stored within multiple resolutions, i.e. level of details, where the image size is halved in each subsequent level until a size of 1×1 is reached.



(a) $a = 22$ (b) $a = 88$

(c) $a = 22$ (d) No depth of field

▲ **Figure 5.8 —** Depth of field (DoF) is used to focus on the object of interest while providing some context. (a), (b) focused on a protein with different aperture values. (c) focused on the lower microfilament. (d) original scene without DoF.

▼ **Table 5.1 —** Performance measurements of image-space depth of field (DoF). For filtering of the mipmaps, a 5 × 5 Gaussian kernel is used. The ShaderX approach was implemented according to Scheuermann and Tatarchuk [2004]. The hardware configuration is given in Appendix B.

Technique	MAPK atomic vis [ms]				Pores [ms]
Number of samples	8 taps	12 taps	16 taps	32 taps	12 taps
Without DoF	180.213	—	—	—	2.762
Mipmap DoF	181.258	181.587	181.818	182.749	4.125
Mipmap DoF (filtered)	181.422	181.851	182.05	182.983	4.431
ShaderX implementation	181.686	182.382	255.754	277.008	4.684

traffic is necessary for the texture fetches, while in the novel approach most of the texture fetches are performed at smaller mipmap levels and, thus, less memory bandwidth is required.

5.2 Schematic Cellular Visualization

The schematic visualization approach employs geometric objects and glyphs to display the simplified cellular components. The purpose of this schematic representation is to provide a visualization identical to the properties of cellular simulation, which relies on geometric shapes like spheres and cylinders. The depiction of the crowding in the cell, together with highlighted proteins of interest, is close to reality and shows the stochastic distribution of the molecules. Especially in signal transduction, where the low number of molecules leads to high levels of noise, this representation is superior to simple plots of the distribution profile. Due to the high spatio-temporal resolution of the simulation, the detailed visualization provides a better understanding of the cell and signal transduction processes.[2]

The visualization includes the plasma membrane, the nucleus, the filaments of the cytoskeleton, and the signaling molecules. In addition, static obstacles can also be included, which are often spherical and hinder diffusion. Cylinders representing the cytoskeleton are the largest objects inside the plasma membrane and often responsible for most of the occlusion. Displaying the cylinders increases the visual complexity tremendously, but exactly these cytoskeletal filaments support spatial perception. Without the filaments there remains only a point cloud of particles and additional depth cues are necessary (cf. Section 5.1.3). An optional color gradient with cool-warm shading can be used for the shading of the cytoskeletal filaments. The color is chosen depending on the relative distance of the point from the nucleus. In this way, the user is always aware of the camera orientation.

Non-static molecules are rendered in their approximated, spherical shape at the particle positions. The particles are colored according to their respective molecule type. In addition, the global visibility of a molecule type can be toggled, affecting the display of the individual particles. Some particles might also be too small to be clearly visible at the desired resolution of the cell. Hence, a selected class of proteins can be scaled up in the visualization by a user-defined scaling factor. During the interactive playback of the simulation results, linear interpolation between the particle positions of two time steps is used, resulting in a smooth animation.

Proteins can traverse large parts of the cell leaving long trajectories behind them. For the visualization of such trajectories, the particle positions of different time steps are joined by thin illuminated cylinders instead of line segments. Line segments, typically

[2] **Parts of this section have been published in**: M. Falk, M. Klann, M. Reuss, and T. Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis 2009)*, pages 169–176, 2009.

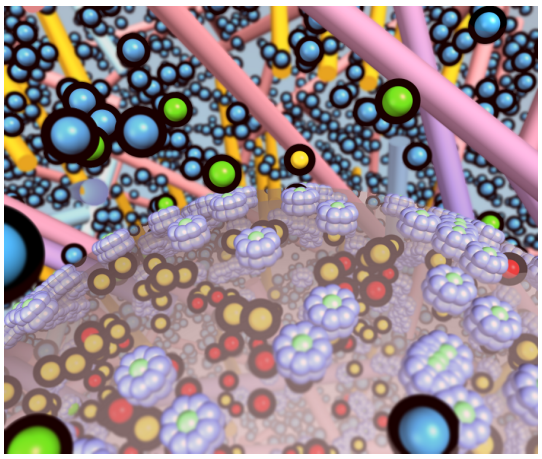
used for the illustration of trajectories, impede the correct estimation of length and direction of a single line segment in the spatial domain due to constant line widths and missing occlusion. In contrast, thin cylinders allow perceiving the direction of single steps by means of lighting. Depth perception is enhanced by the inherent perspective miniaturization and self-occlusion of the trajectories. Markers along a trajectory indicate the direction of movement of the protein on a particular segment. Small arrow tips are added where space is available, i.e. where the segment exhibits a certain minimum length (cf. Figure 5.10).

5.2.1 Technique

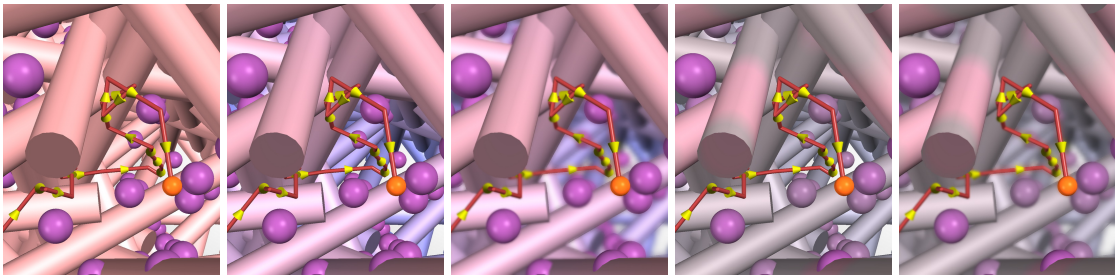
In the schematic visualization, GPU-based glyph rendering is employed for the visualization of all spherical molecules. Thereby, only the particle positions and the respective molecular radii have to be uploaded into vertex buffers stored in GPU memory for efficient rendering. The position data in the vertex buffers is updated in accordance with the selected time step. To allow for interpolation between positions, the positions of the current time step and the next subsequent step are stored on the GPU. For each particle to be visualized, a single point is rendered. The vertex shader estimates the silhouette size of each particle and adjusts the point size to cover the complete silhouette. Every fragment covered by the enlarged point is then evaluated in the fragment shader resulting in a spherically rendered glyph (cf. Section 2.5.2). If halos are to be rendered for the individual particles, the particle radii are enlarged by the radius of the halo. The glyph calculations in the fragment shader are adjusted to consider the annulus of the halo surrounding the spherical particle. The halo depth is computed similar to the depth of the depth-dependent halos proposed by Tarini et al. [2006].

In the fragment shader, the shading and illumination computations take place as well as the optional desaturation for depth cuing. The particle trajectories of selected particles are precomputed upon user request and the complete trajectory is stored also in GPU memory. The individual points of the trajectory are connected by cylindrical glyphs, i.e. a small quad is rendered for every segment and the cylinder equation is evaluated in the fragment shader similar to the spherical glyphs. Markers are added to the segments in the same way by solving an equation for capped cones.

The filaments of the cytoskeleton are represented by cylinders, which are explicitly stored as a triangle mesh. Early testing with a glyph-based rendering revealed that the performance with a mesh-based approach is more suitable. This is due to the characteristics of the cytoskeletal filaments. The filaments occur everywhere in the cell and they are large when compared with the diffusing particles. Thus, they cover large parts of the viewport resulting in a high load during fragment processing. The mesh containing all filaments is stored in a vertex buffer. To reduce the memory footprint for cell models containing several cytoskeletal filaments, a second approach employing instances is available. Instead of storing the geometry of every single filament, only one template is created and stored on the GPU. The instantiated template is transformed by scaling, a rotation, and a translation for each filament. In both approaches, the color gradient for the cool-warm shading is



▶ **Figure 5.9 —** Schematic visualization of the nuclear pore complex scenario. The nuclear envelope of the nucleus is rendered semi-transparent to reveal the molecules otherwise hidden inside.



(a) (b) (c) (d) (e)

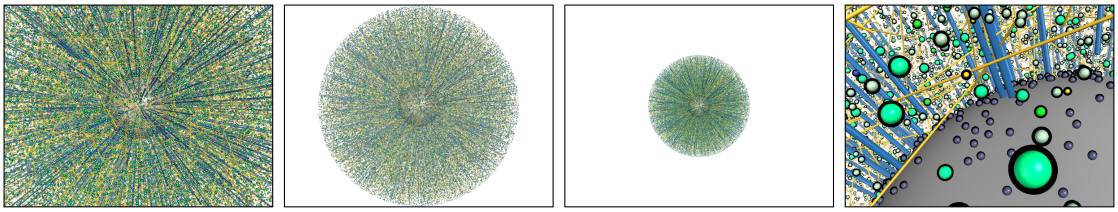
▲ **Figure 5.10 —** Depth cues in the schematic cellular visualization. A signaling molecule (orange) is moving inside the cell between obstacles (purple) and the cytoskeleton (rose to light blue). (a) without depth cues. (b) color gradient on the cytoskeletal filaments. (c) color gradient and DoF. (d) desaturation and color gradient. (e) combined depth cues.

computed during rendering in the fragment shader. On the latest graphics hardware, the glyph-based approach with a tight-fitting bounding geometry or the recently introduced tesselation shader (cf. Section 2.4.1) can also be used for the rendering of the cytoskeleton.

5.2.2 Results

The schematic visualization renders the data resulting from the simulation reusing the originally modeled geometrical shapes. In Figure 5.9, one time step of the nuclear pore complex scenario is depicted with the schematic visualization. Halos are used to ease the distinction of individual molecules especially in the plasma membrane, where the blue molecules are prevalent. The molecules inside the nucleus become visible by rendering the nuclear envelope semi-transparently. DoF is used to focus on the yellow and green molecules in the center of the scene, thereby pointing out the different scales in the intracellular environment.

The different possible depth cues in combination with the trajectory of a single molecule are shown in Figure 5.10. Without any depth cues and relying only on the perspec-



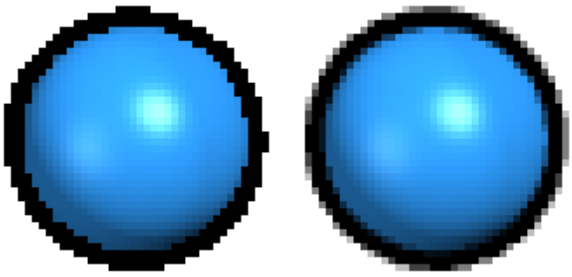
(a) (b) (c) (d)

▲ **Figure 5.11** — Camera settings for the performance measurements of the schematic visualization. (a) zoomed in onto the cell to just cover the whole viewport. (b) entire cell. (c) entire cell covering half the viewport. (d) closeup of the interior of the cell with halos around the molecules.

tive miniaturization and occlusion, depth estimation is very difficult in the crowded environment (Figure 5.10(a)). Applying cool-warm shading onto the filaments of the cytoskeleton already eases depth perception as shown in Figure 5.10(b). The effect of DoF (Figure 5.10(c)) is best suited to highlight, e.g., a single molecule, while providing some contextual information in the background and foreground. In addition to the color gradient along the cytoskeletal filaments, desaturation can be used to enhance depth perception. In Figure 5.10(d), the desaturation does not depend on the scene depth but is instead adjusted to the radius of the CoC of the DoF, i.e. high saturation is enabled at the focal point and less saturated colors are used in the unfocused regions. This allows to guide the viewer's attention to a specific region similar to DoF. Finally, Figure 5.10(e) shows the combination of all previously described depth cues.

Measurements were taken for four distinct camera positions and different configurations of the visualization at a viewport resolution of 1600×1200. The last time step at $t = 1.8$ s of the simplified MAPK pathway scenario including 70 000 molecules was used for measuring (cf. Section 4.5.1). In the first setting, the camera is positioned, so that almost the entire virtual cell is visible but the corners of the viewport are also covered (Figure 5.11(a)). The cell covers the full height of the viewport (Figure 5.11(b)) and only half the viewport height (Figure 5.11(c)) in the next two configurations. The last setting allows a glance into the interior of the cell with the nucleus nearby (Figure 5.11(d)).

In Table 5.2, the performance measurements are given for the simplified MAPK pathway example. The same scene was rendered with different settings including protein-only and skeleton-only visualizations, the plain rendering without any depth cues, and the combination of halos and desaturation. The schematic visualization of the 70 000 molecules is highly interactive at frame rates between 240 and 600 fps. Tests with the cytoskeleton instancing instead of the fully modeled triangle meshes resulted in less than half the performance yielding only between 157 and 312 fps for the skeleton-only setting. The time spent for depth cues, halos and desaturation, is almost negligible with 0.78 ms when the cell covers the whole viewport. However, the effort needed for such depth cues depends on the number of rendered molecules and their size. This can be seen in the fourth camera setting, where a smaller number of molecules is visible and the computation of the depth cues consumes a total of 0.35 ms.



(a) (b)

▶ **Figure 5.12 —** Multisampling can be used to improve the image quality of ray-casted glyphs. (a) staircase artifacts are evident without multisampling. (b) fragment shader called for every sample during multisampling.

Multisampling of Glyphs

To improve the overall image quality, anti-aliasing techniques can be employed. Due to the rasterization and the possibly low resolution of the output device, staircase artifacts at the edges of polygons might appear. With supersampling, the scene is rendered at a higher resolution and downscaled afterward to the desired output resolution, thereby reducing aliasing artifacts. Using multisampling, these artifacts can be reduced with less computational effort than is necessary for supersampling.

During rasterization, the rendered geometry is rasterized into multiple sample buffers with slight offsets, where both color and coverage values are stored. The coverage is defined by the part of the fragment that is covered by the geometry. The fragment shader is called just once per pixel, but the depth test is performed for each sample buffer yielding smooth edges at the border of the rendered objects. The approach, however, works only for points, lines, and polygons. When this approach is applied to ray-casted glyphs, aliasing artifacts still exist since the fragment shader cuts away the outer smoothed parts of the rendered points so that only the spherical silhouette of the glyph remains. To yield an anti-aliased glyph, the fragment shader has to be called several times per pixel. The OpenGL extension `GL_ARB_sample_shading`[3] enables this functionality and the fragment shader is called explicitly for each sample position. This greatly reduces aliasing artifacts at the glyph boundary and also inside the glyph, as can be seen in Figure 5.12. Since the fragment shader is called several times during multisampling, the rendering times increase when

[3] http://www.opengl.org/registry/specs/ARB/sample_shading.txt

▼ **Table 5.2 —** Measurements of the schematic visualization with the MAPK scenario. Timings per frame are given in milliseconds [ms]. The hardware configuration is given in Appendix B.

Setting	Proteins	Skeleton	Plain	Halos	Color cues	Halos & cues
Covered	1.531	3.015	3.733	4.141	4.161	4.511
Full height	1.238	2.01	2.654	2.916	2.865	3.117
Half height	0.928	1.151	1.617	1.736	1.721	1.812
Closeup	1.488	1.883	2.217	2.502	2.32	2.566

compared to the original, aliased scene. Depending on the number of rendered glyphs, the required time per frame increases by a factor of 1.5 to 2.4.

5.3 Atomistic Visualization of Mesoscopic Cells

The atomistic visualization, which is described in the following, is based on the work by Lindow et al. [2012]. The original approach is optimized and two acceleration approaches are proposed to visualize the cellular environment with 25.42 billion, i.e. 10^9, atoms interactively. The extended visualization technique is applied to the results of the mesoscopic simulation of the simplified cell model to enrich the schematic visualization by details on the atomic scale. This can help both domain experts and laymen to observe the detailed function of cellular processes and gain a better understanding of cellular dimensions and complexity. While atom-based simulations typically range up to tens of millions of atoms, mesoscopic simulations of a simplified cell model, as the one described in Chapter 3, can easily reach tens of billions of atoms, if shown on an atomistic level. This large number of atoms, however, demands for sophisticated visualization techniques as it is pushing the rendering capability of current GPUs to its limits. With occlusion culling and deferred shading, Grottel et al. [2010] were able to interactively render data sets with about 100 million particles. Such acceleration techniques, however, are only beneficial for dense particle data with inherent occlusion. That is why this approach is not applicable to the sparsely populated interior of a cell and other means of acceleration are necessary.[4]

In 2007, Lampe et al. [2007] proposed an approach to render proteins by assembling individual amino acids on the GPU. The amino acids are considered to be static and, hence, a protein can be compiled of amino acid chains by translation and rotation. The atom positions are stored in textures for each amino acid chain and the atoms are instantiated with the geometry shader to generate the proxy geometry for ray casting. The approach is, hence, limited by the number of primitives that the geometry shader can emit. Lindow et al. [2012] recently presented a technique, which overcomes this weakness. Their data originates from electron tomography of cells from which ribosomes and cytoskeletal filaments are reconstructed and subsequently visualized at the atomic scale. Since the data is highly repetitive, i.e. many identical instances of molecules, the whole data set consisting of about 10 billion atoms can be visualized at interactive frame rates.

Similar to the work by Lindow et al. [2012], the following assumptions apply to the results of mesoscopic cellular simulations. First, the scene contains billions of atoms but the number of individual molecule types is low, i.e. several signaling molecules and cytoskeletal molecules feature the same molecular structure. Secondly, molecules are rigid. Hence, the molecular structure has to be stored only once per molecule type, thereby reducing memory requirements drastically. Ray casting is used for high quality rendering of the atomic structures.

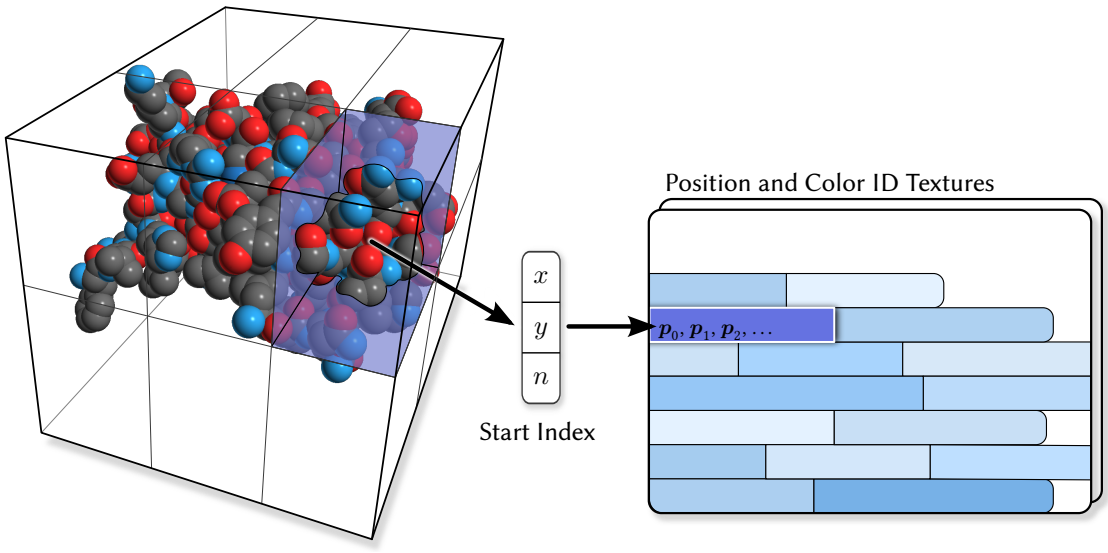
[4] **Parts of this section have been published in**: M. Falk, M. Krone, and T. Ertl. Atomistic visualization of mesoscopic whole-cell simulations. In *Eurographics Workshop on Visual Computing for Biology and Medicine (VCBM)*, volume 2, pages 123–130, 2012.

The information on the atomic structures themselves is obtained from the protein data base (RCSB PDB). Information on the correct assembly of the structures is provided by the RCSB PDB and the respective literature. The RCSB PDB data files provide information on the location and type of atoms as well as conformal information. In the proposed visualization, signaling molecules and the filaments of the cytoskeleton, i.e. microfilaments and microtubules, are replaced with their respective atomic structures. The lipid bilayers of the plasma membrane and the nuclear envelope are not modeled at the atomic scale and remain in their geometric representation. Small proteins can usually be loaded from a single RCSB PDB file. In some cases, however, the file might only contain a part of the protein, which has to be combined with other protein parts. Microtubules are composed of α- and β-tubulin dimers (PDB ID: 1TUB). The dimer is about 8 nm long and is continuously shifted along the microtubule axis reaching 12 nm per turn [Lodish et al., 2007] yielding a hollow structure (cf. Section 2.1.1). For the atomic visualization of the microtubules, two templates are generated that contain the positions of the atoms. The first template consists of 130 tubulin dimers arranged in ten rows resulting in 882k atoms and the second one of twenty rows (1.76M atoms). The template blocks of the microtubules are fitted on the respective filaments of the cytoskeleton from the simulation. Since the elongation of those blocks is small compared to the total length of the filaments, the remaining uncovered part is less than 80 nm and, therefore, negligible. Actin monomers (PDB ID: 3MFP) form a helical structure. One complete turn is built of 28 actin moleculesand the length of one turn corresponds to 77.3 nm. The helix can be constructed either with the transformation provided in the RCSB PDB entry or the model of Holmes et al. [1990]. The two templates of the actin filaments consist of 28 actin molecules (82k atoms) and 56 molecules (164k atoms), respectively.

5.3.1 Technique

Since the molecules do not undergo any internal deformations in the simulation, a single molecular template is sufficient per protein type and, thus, the atomistic models have to be transferred to the GPU memory only once. During rendering, multiple instances of the same molecule are drawn. This allows keeping the amount of data which has to be transferred for each rendering pass from the CPU to the GPU is very low. For each rendered molecule, only a translation and a rotation transformation have to be transferred, which can either be stored in a single matrix or in a quaternion and an additional translation vector. In the vertex shader, the quaternion rotation must then be converted back into a matrix. This computational overhead is compensated by the fact that less data has to be transferred to the GPU, i.e. only 7 values for a quaternion and translation compared to 12 values for a matrix.

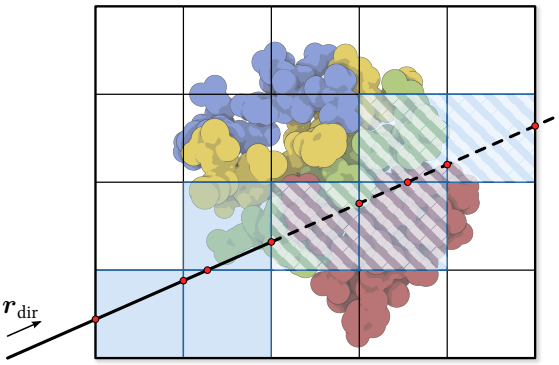
The atoms of a template molecule are embedded into a uniform grid (see Figure 5.13). This grid is used for achieving spatial subdivision to reduce the number of intersection tests between viewing rays and the atoms. All atoms of a molecule are inserted into this grid. A box-sphere intersection test is used to determine all cells of the grid with which an atom intersects. The size of the grid cells was chosen with respect to the atom



▲ **Figure 5.13 —** Data structures of the atomistic visualization. The protein (insulin, PDB ID: 1RWE) is embedded into a uniform 3D grid, where each grid cell refers with an index to a consecutive list of atoms intersecting with this cell. The lists containing the atomic information, i.e. atom position, radius, and coloring, are stored in two 2D textures.

size and, therefore, a voxel size of 4 Å, i.e. 0.4 nm, is used. Thus, it is ensured that a single atom is contained by at most eight grid cells. The grid data structure proposed by Lagae and Dutré [2008] is used to store the data of the uniform grid. Each cell of the grid is mapped onto one voxel of a 3D texture. The atomic data, i.e. atom position, radius, and color IDs, are stored in two additional 2D textures. The first texture contains the atom position and its radius represented in 32-bit floating point values. The color identifiers, e.g., atom type, chain ID, or strand ID, are stored in the second texture with up to four channels. This additional data is accessed via a 2D index (x, y) stored in the 3D grid texture. In addition to the 2D index, each voxel also contains the number of atoms n in this cell. Imprecision in indexing can be avoided by using a 16-bit integer format for the 3D texture, thus allowing precise addressing of textures up to 32k × 32k texels.

To initiate the ray casting of the atoms, only the grid's bounding box of each instance is drawn. By rendering only the back faces of the bounding box and computing the corresponding position on the front side, the ambiguity between front and back faces is avoided. The fragment shader computes the view ray through each fragment covered by the bounding box and traverses the grid cells front to back. The grid traversal follows the idea of Amanatides and Woo [1987] for a fast and discrete ray-voxel traversal. The individual ray-sphere intersections for the atoms are consequently computed per grid cell. As soon as at least one atom is hit, the grid traversal can be stopped. Figure 5.14 illustrates the grid traversal for a single ray. It is, however, important to compute all intersection in one grid cell to obtain the closest intersection since the atoms are not ordered.

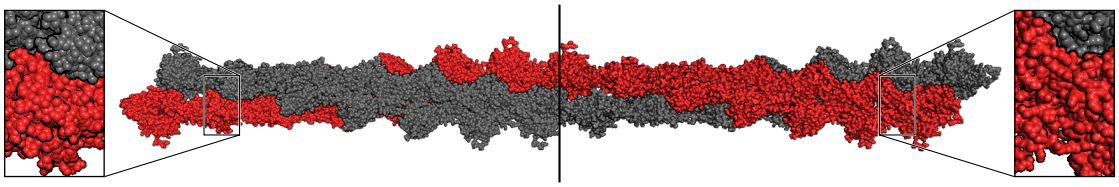


▶ **Figure 5.14** — Atomic ray casting of the uniform protein grid. The grid cells traversed by the ray $\boldsymbol{r}$ are marked blue. Once an intersection was found, the grid traversal is terminated and the remaining grid cells are neglected (hatched cells).

The signaling molecules of the cellular particle simulation are all represented by spheres. In the simulation, rotational diffusion is not explicitly modeled, but instead incorporated into the reaction rates. To account for the random rotation of proteins in the visualization, the rotational diffusion is added by applying random rotations onto each molecule when a particular time step is loaded. For consecutive time steps, the rotational perturbation is incrementally applied to ensure a smoother appearance.

The effect of OSAO [Grottel et al., 2012] can be included without any additional computations or preprocessing since the uniform grid has a low resolution and already contains the number of atoms per cell. During rendering, the number of atoms near the computed intersection is accessed using a small offset. The scaled result can directly be used as ambient occlusion factor, which is then passed on to the deferred shading pass. Deferred shading [Saito and Takahashi, 1990] is used to postpone the illumination and shading calculations, except for the OSAO, into a separate step. This is advantageous because the calculations are only performed for visible fragments instead of every single ray-sphere intersection, thereby reducing the computational load. In addition, various postprocessing techniques can be applied in image space, including SSAO and depth-dependent silhouettes. Due to the small sizes of the atoms, high frequencies occur between adjacent normals of distant objects, which might be emphasized by specular highlights. Grottel et al. [2010] and Lindow et al. [2012] propose normal correction schemes to smooth out such high frequencies. In this thesis, the analytical normal is only used for atoms near the camera, whereas an approximated normal is used for objects that are further away. The approximated normal is the normal of the center point of a quadratic Bézier surface over the current point and its neighbors. Linear interpolation is employed to obtain a smooth transition between exact and approximated normal depending on the camera distance. In Figure 5.15, the normal approximation is applied to the right half of a microfilament. Besides the smoother, more surface-like appearance, the normal approximation also accentuates the roughness of the structures.

In the following, two approaches are presented to improve the rendering performance of the atomistic visualization. The first method called “depth culling” introduces an additional depth buffer to avoid unnecessary computations in the fragment shader. The second approach makes use of the hierarchy implicitly given by embedding the atomic structure into the 3D grid.



▲ **Figure 5.15 —** Normal approximation applied to a microfilament consisting of 28 actin molecules. The normal approximation (right) enhances the global structure compared to the shading with exact normals (left).

Depth Culling

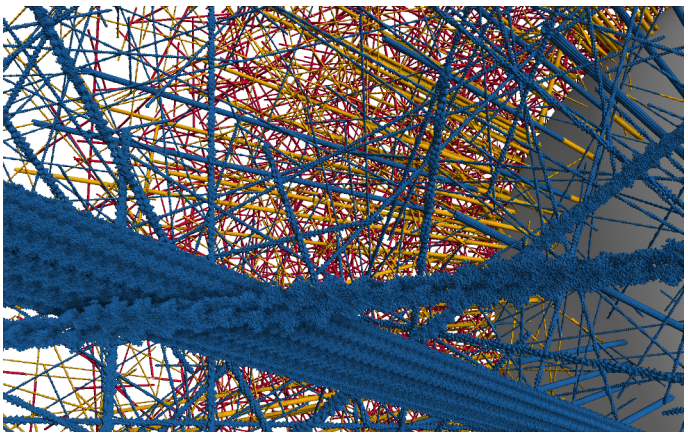
GPUs provide the *early z-test* during rasterization to reduce the rendering workload for scenes with high depth complexity or massive overdraw (cf. Section 2.4.1). Since the cellular environment is crowded with proteins and cytoskeletal filaments it exhibits a high depth complexity and the atomic visualization would benefit from the *early-z test*. However, the *early-z test* is disabled by GPU as the fragment depth has to be manipulated in the fragment shader. For correct molecule-molecule intersections the depth correction of found ray-sphere intersections is necessary and, in case the ray does not hit any atoms, the fragment shader must discard fragments.

To emulate the *early z-test*, the depth of the closest intersection is stored and reused in the fragment shader to perform an early reject. If the stored depth value is smaller than the depth of the front side of the bounding box, the computation in the fragment shader can safely be discarded since the current fragment is occluded. In case the front position is closer, the grid traversal takes place and if an intersection is found, the resulting depth is stored into a second texture. The two textures with the depth values are swapped in a ping-pong fashion after n molecule instances have been drawn. The regular depth buffer is still necessary to ensure the correct depth ordering of the n instances within one pass.

The depth information is combined with the normal and the color ID resulting from the ray-sphere intersections to avoid a second render target, which would slow down the rendering. By using integer color IDs, the scaled third component of the normal can be stored together with the color ID . Hence, it is possible to store the normal, the color ID, and the closest depth in a single texture. In the subsequent deferred shading, both result textures are sampled and the texture with the smaller depth value is chosen.

Hierarchical Ray Casting

In the atomic visualization, many atoms will cover at most one pixel of the final image. This also applies to signaling molecules that are located far away from the camera. The low pixel coverage of atoms can be exploited to avoid the costly the grid traversal. Before the ray-intersection tests take place in a grid cell, the voxel size is compared with the pixel size corresponding to the entry point of the ray. The pixel size is obtained for the respective depth with the intercept theorem from the viewport height in pixels and the



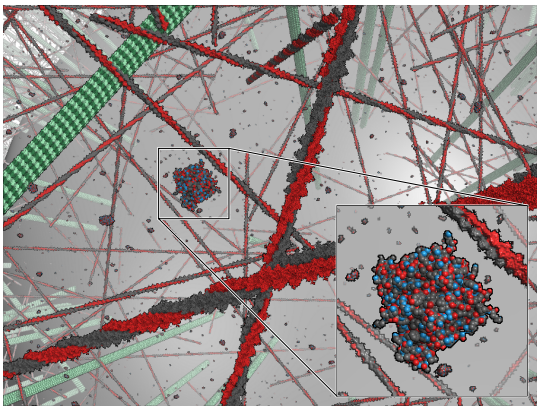
▶ **Figure 5.16 —** Hierarchical ray casting provides full detail (blue) in the foreground. If the pixel size is larger than a grid cell, only filled grid cells (yellow) are considered. Molecule bounding boxes covering only parts of a pixel are drawn in red. Pixel sizes are exaggerated, so that the effect becomes visible.

field of view. If the pixel is larger than the voxel, i.e. the voxel covers only a part of the final pixel, the ray-sphere intersections can be omitted because individual spheres will not be discernible. Instead of the intersection test, only one texture lookup is made to determine whether the voxel is empty or not resulting in a binary voxelization. As the grid size was chosen with respect of the atom sizes and the voxel contains at least one atom, the chance of one atom being hit by the ray is high and, hence, the grid traversal can be stopped for non-empty voxels. In this particular case, a special coloring scheme has to be employed since no intersection tests are performed. Using the color of the first atom entry has the advantage that coloring according to chain, strand, or instance ID is still possible and also visible. Other possible color schemes could apply a precomputed averaged color per grid cells or just a single color, which is predefined per molecule instance.

This approach is not only applicable to grid cells but also to the entire bounding box of the molecule. The approach then also prevents molecules from becoming invisible due to sizes smaller than a pixel. The combination of the regular grid traversal and the consideration of pixel sizes for both, molecular bounding box and individual grid cells, results in a hierarchical ray casting. In Figure 5.16, the different levels of detail are highlighted when hierarchical ray casting is applied. Since the normals of far-away atoms are approximated during deferred shading, the binary voxelization of the grid does not show up in the final image.

5.3.2 Results

For performance measurements, the atomic visualization is applied to the generalized MAP3K pathway scenario (cf. Section 4.5.2). The general pathway is specialized into a mitogen-activated ERK pathway by substituting the generalized MAPK molecules with specialized ones. In the following, the four digit alpha-numeric expressions in brackets denote the corresponding RCSB PDB entries that are used for the underlying atomic structure. The upstream part of the pathway (MAP3K) is instantiated with B-Raf (PDB ID: 3PPJ). MEK1 (PDB ID: 1S9J) replaces MAP2K on the second tier. Inactive and active, i.e. phosphorylated, MAPK molecules are substituted with ERK1 (PDB ID: 1ERK). The counteracting phosphatases are modeled with MKP3 (PDB ID: 1MKP). In addition, all filaments of the cytoskeleton are replaced by segments of either microtubules (PDB ID:



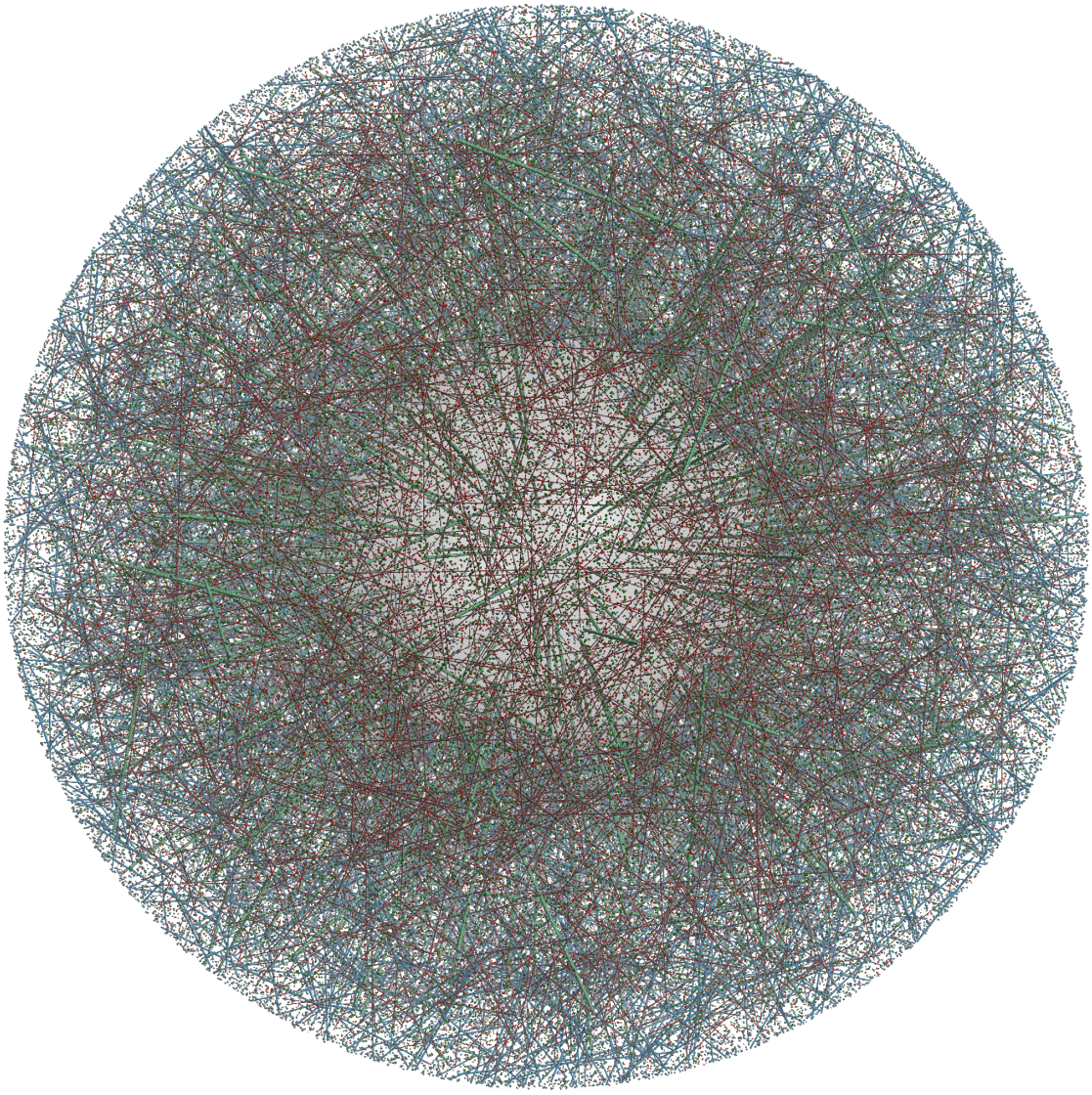
◀ **Figure 5.17 —** Close-up view of the ERK model with the atomistic visualization showing the cytoskeleton and signaling proteins. Individual atoms are discernible for molecules near the camera.

1TUB) or microfilaments (PDB ID: 3MFP). The number of elements in the model and the respective atom counts for the ERK pathway example are shown in Table 5.3. In Figure 5.17, a close-up view of the region near the nucleus reveals the atomic structure of a signaling molecule. Individual atoms are, however, distinguishable only for molecules close at the camera because of the huge differences in scale inside the cell. The cell with an outer diameter of 10 µm is five orders of magnitude larger than a single carbon atom (0.34 nm), which is a basic constituent of proteins. The atomic visualization of the whole data set is depicted in Figure 5.18 comprising over 25 billion atoms in total.

The measurements of the ERK pathway were taken with four distinct camera settings similar to those depicted in Figure 5.11. The viewport resolution was set to 1600 × 1200 pixels and the 2D textures are swapped after rendering 4096 molecules, if depth culling is enabled. The results are given in Table 5.4. For full viewport coverage, depth culling improves the performance by 6 % whereas the hierarchical ray casting results in an increase of 61 %. By combining both optimizations, a speedup of about 80 % was achieved. Depth culling is most beneficial if the rendered molecules or other objects, like the nucleus, occlude large parts of the viewport and thereby increase the depth complexity. This is confirmed by the measurements of the close-up view. Hierarchical ray casting has only a negligible effect on the rendering performance, which is improved considerably by depth culling. The combination of both optimizations results in slightly higher timings since the remaining, visible structures are not effected by voxel culling of the hierarchical ray casting.

▼ **Table 5.3 —** Parameters of the components of the ERK model.

Type	# Elements	Length [µm]	# Instances long	# Instances short	# Atoms
Actin	7500	9873	62 051	3622	14.34×10^9
Microtubules	800	1303	7954	378	10.44×10^9
ERK	246 000	—	—	—	0.59×10^9
Total					$25\,421\,804\,076 = 25.42 \times 10^9$



▲ **Figure 5.18** — Atomistic visualization of the ERK pathway scenario. The data set comprises about 25 billion atoms in diffusing molecules and cytoskeleton.

▼ **Table 5.4** — Measurements of the atomic visualization of the ERK pathway. Timings per frame are given in milliseconds [ms]. The hardware configuration is given in Appendix B.

Setting	No opt.	Depth Culling	Hierarchical	Combined
Viewport covered	538.503	506.073	334.336	301.114
Full height	571.102	506.842	262.674	235.294
Half height	301.205	288.6	62.461	47.76
Closeup	253.807	141.143	253.55	142.816

Table 5.5 shows the results for different, isolated molecules. Both optimization techniques are disabled since only a single bounding box is rendered for each structure. The differences in the rendering times are partly due to the number of atoms but are also affected by the viewport coverage. Both filament types are elongated and thus cover less pixels than, e.g., the viral envelope, the ribosome, or insulin, which all have a sphere-like shape in common. The normal approximation requires only fraction of a millisecond but significantly improves the image quality by enhancing the shape of the surface and by reducing artifacts due to high frequencies (cf. Figure 5.15). The overhead of the OSAO is insignificant when compared to the image-space approach of SSAO, because it is only one additional texture lookup if a ray-sphere intersection was found.

5.4 Microscopic Visualization of Cells

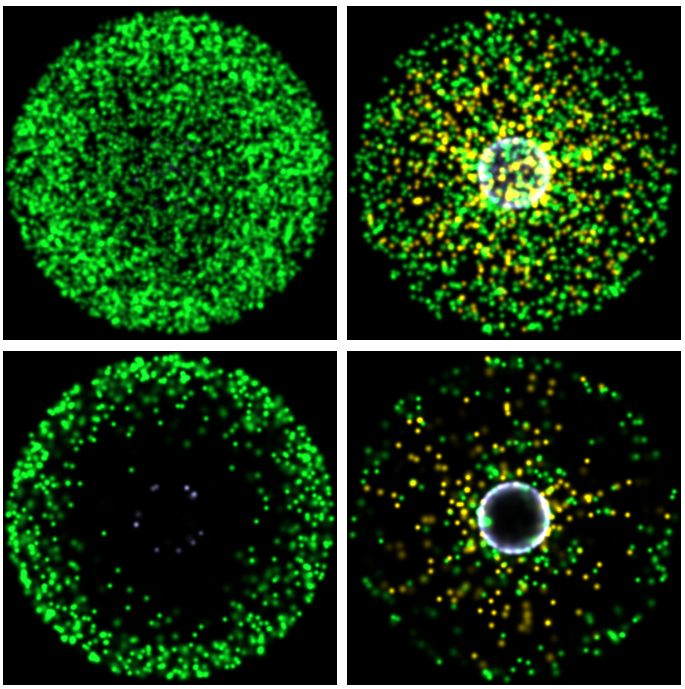
The goal of this visualization approach is to create images familiar to biologists and to facilitate the comparison between *in silico* simulations and *in vivo* or *in vitro* experiments. As most images from experiments are made with microscopes, the need arises to represent simulated data in a similar manner. Images generated by CLSM (cf. Section 2.1.3) served as inspiration for this visualization technique, since both molecule counts and locations are captured in a single image.[5]

Rendering the precise molecule positions from the simulation results in points, which do not effect proximal points unless alpha compositing or more sophisticated techniques are employed. In the proposed approach, small camera-aligned quads are used for each particle position to overcome this issue. Due to the enlarged area of the particles, a higher coverage of the image plane is accomplished. A radial intensity distribution is applied to each particle to mimic the Gaussian intensity profile of the laser obtained in CLSM. The emitted color of each molecular species is defined by the user. In combination with an additive blending function the resulting brightness looks much like light emitted by

[5] **Parts of this section have been published in**: M. Falk, M. Klann, M. Reuss, and T. Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis 2009)*, pages 169–176, 2009.

▼ **Table 5.5 —** Measurements of the atomic visualization of molecule structures. Timings per frame are given in milliseconds [ms]. The hardware configuration is given in Appendix B.

Structure	# Atoms	No Effects	Normal Approx.	OSAO	SSAO
Microtubule	1 765 140	5.929	6.035	6.189	33.738
Viral envelope (1SVA)	958 980	16.793	17.027	17.467	34.855
Microfilament	163 688	2.444	2.444	2.482	27.442
Ribosome (2WD*)	147 236	9.937	10.08	10.401	30.358
Insulin (1RWE)	823	3.623	3.793	3.909	32.165



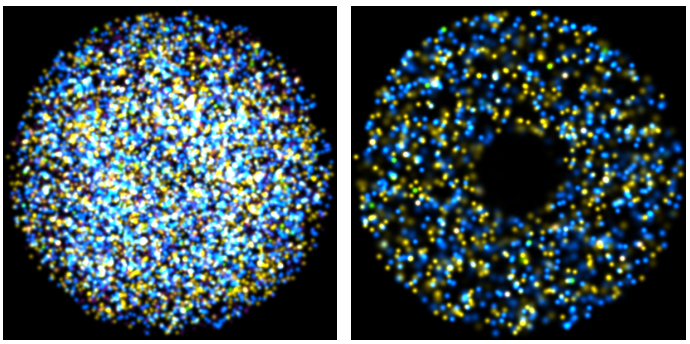
▶ **Figure 5.19 —** Microscopic visualization of the simulation results of the simplified MAPK pathway at $t = 1.8\,\mathrm{s}$. On the left, molecules move only by diffusion whereas transport by motor proteins is enabled on the right. Diffusing MAPKp molecules (green) change to yellow when attached to the cytoskeleton and turn into cyan when adsorbed at the nucleus. The focal plane is enabled in the bottom row.

fluorescent markers. The overall appearance is slightly blurred like in the experimental images. Only tagged, i.e. user-selected, molecular species are rendered and all other molecule types are discarded. The final result is an image with high similarity to the experimental ones, thus allowing an easy comparison.

Without further adjustments the microscopic visualization generates images of the whole cell incorporating all selected molecules (Figure 5.19, top row). This corresponds to the volume rendering of the reconstructed 3D cell from CLSM images. If only a single slice or cross-section should be visible, two potential approaches exist. In the first approach, only the visible particles are rendered. This can be handled either by preselecting the visible particles or by using the clipping mechanism of the GPU. The second approach relies on a more advanced concept featuring high similarity with the optical phenomena in CLSM. A focal plane is positioned at the desired depth like in the experiment. The point size of a particle is proportional to the distance between the particle position and the focal plane. Hence, particles that lie not in the focal plane become more and more blurry. The intensity of the particle is reciprocal to the point size to yield a constant emitted amount of light per particle, i.e. small points appear brighter whereas large points are barely visible. In the bottom row of Figure 5.19, the focal plane is positioned at the center of the nucleus resulting in a cut-away view.

5.4.1 Technique

Point primitives are used at the positions of the particles for the rendering. The particle positions are stored in same vertex buffer that is also used by the schematic visualization. Without the focal plane, the vertex shader uses a constant point size for every particle



(a) (b)

◀ **Figure 5.20 —** Microscopic visualization of 10 000 randomly distributed particles. (a) whole cell without a focal plane. (b) with a focal plane at the nucleus.

and adjusts the point size. The respective fluorescent colors are passed on to the fragment shader. In the fragment shader, the Gaussian intensity profile is evaluated for each fragment covered by the enlarged point. The Gaussian function is computed directly in the fragment shader instead of using a 1D or 2D texture lookups to avoid sampling artifacts due to over- and undersampling. The result of the fragment shader is then composited with additive blending into the frame buffer.

When the focal plane is enabled, the vertex shader determines the point size dynamically and a weighting factor is computed. First, the distance of the particle toward the focal plane is determined. The depth attenuation along the viewing direction, which is oblique to the focal plane, is approximated with a Gaussian function where the standard deviation determines the depth resolution of the visualization. A small standard deviation leads to a narrow but sharp cross-section whereas a large value results in a broader slab that is more blurry. The depth attenuation is used to adjust the intensity of the point and the inverse is used for scaling the point size. For optimization reasons, points with an overall low contribution, i.e. a depth attenuation value of less than 0.005, are discarded. In addition, the maximum point size is set to 256×256 to limit the required fill rate. If a particle exceeds this maximum size, the intensity is adjusted accordingly and the maximum point size is used instead. The depth attenuation factor is passed to the fragment shader, where it is used to scale the intensity per fragment.

5.4.2 Results

The microscopic visualization has the ability to generate simple but informative images that enable the user to easily grasp the molecule distribution inside the cell. Figure 5.19, which shows a single time step of the simplified MAPK pathway, is such an example. On the left it is immediately evident that the diffusion of molecules is hindered by the phosphatases since the space around the nucleus is empty. Only a few molecules have been able to reach the nucleus, which is indicated by a few cyan dots. After the same amount of time has passed, a totally different image is created when motor proteins are transporting the signaling MAPKp molecules toward the nucleus. The overall distribution seems to be more uniform in the cytoplasm.

The performance of the microscopic visualization was measured for different numbers of particles, which are randomly distributed inside the plasma membrane. The data resulted from the performance benchmarks of the simulation described in Section 4.4.2. Figure 5.20 depicts the results of the microscopic visualization for 10 000 particles. Table 5.6 shows the results of the measurements at a viewport resolution of 1600 × 1200 performed on the hardware given in Appendix B. The additional calculations, which are necessary for the depth attenuation with the focal plane, result in a slight overhead. Much of the effort is spent on the rasterization and the filling of the large points. The times per frame given in Table 5.6 are, therefore, only valid for the initial point sizes of 5 nm for the whole cell and 2 nm with the focal plane enabled. Larger point sizes would lead to an increase of the fill rate and, therefore, result in lower performance. In contrast, using smaller point sizes would yield higher frame rates.

5.5 Continuous Distributions from Discrete Particles

Particle-based models are well suited to simulate non-uniform concentrations and low particle numbers (cf. Section 2.2.2), but exactly these properties complicate the visualization. When particles resulting from such an simulation are directly visualized, e.g., as glyphs, global effects are barely visible. It is very difficult to see differences in the particle distributions or to estimate local densities. Sometimes it might be possible to detect those effects through visual exploration and interaction. However, a more global, volumetric visualization, generated from the given particle set, would allow to grasp the density distributions more easily. The global density ρ is obtained by

$$\rho = \frac{\sum_j V_j}{V_{\text{ref}}}, \tag{5.6}$$

where V_j denotes the volume of particle j and V_{ref} is the enclosing volume. However, the density ρ no longer reflects the original distribution of the particles due to averaging and local densities might be more meaningful. The particle density can be approximated locally by computing local densities for small sub-volumes of the enclosing volume. The computation of local concentrations for sparse particle data often results in spikes of high concentrations at the exact particle positions and large empty regions around them, where

▼ **Table 5.6 —** Measurements of the microscopic visualization in the scaling scenario. Timings per frame are given in milliseconds [ms]. The hardware configuration is given in Appendix B.

Particles	Const. intensity	Focal plane
10 000	5.388	6.101
50 000	25.376	28.521
100 000	50	56.475
500 000	247.647	279.174

the density is zero. Such local spikes can be reduced by either increasing the number of particles or by enlarging the sub-volumes. The number of particles is, however, usually determined by the experimental data or the simulation and larger sub-volumes would reduce the spatial resolution. To retain a high spatial resolution and obtain a smooth density distribution at the same time, a novel approach using a radial-symmetric kernel function is proposed.[6]

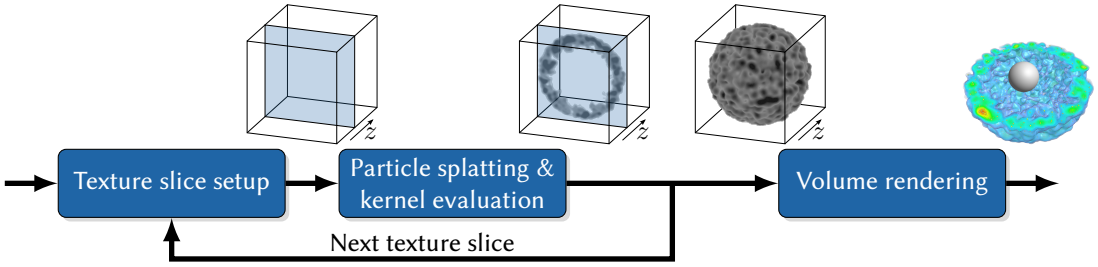
Instead of counting the particles per sub-volume, the contribution of a single particle to the local density is spread over a larger area, which is similar to the microscopic visualization described in Section 5.4. The density contribution of every particle is accumulated in a scalar volume. After considering all particles, this volume represents a density map of the particles. The density profile of a single particle is approximated with a radial-symmetric kernel function $\mathrm{W} \in C^2[0, h]$ with the support h. The local density $\bar{\rho}$ at an arbitrary position $\boldsymbol{p}$ is computed by considering the contribution of all neighboring particles j

$$\bar{\rho}(\boldsymbol{p}) = \sum_j m_j \, \mathrm{W}(|\boldsymbol{p} - \boldsymbol{p}_j|, h), \tag{5.7}$$

where $\boldsymbol{p}_j$ is the position of particle j and m_j its mass. By normalizing the filter kernel with $\int \mathrm{W}(\boldsymbol{p}) \, \mathrm{d}\boldsymbol{p} = 1$, mass preservation is ensured. The support h is user-adjustable and defines the locality of the density computation. By scaling the support, the continuity inside the density volume can be adjusted. With a small support, the density volume reflects the exact particle positions, whereas using a larger kernel results in a more continuous volume. The advantage of this approach is that the support of the filter kernel directly correlates with the error of the density approximation. The particle data is embedded into a 3D uniform grid in order to generate a density volume enclosing the whole particle data set. Equation 5.7 is evaluated for every cell of the grid and the resulting 3D scalar field is visualized with DVR techniques (cf. Section 2.5.3).

When the density volume is visualized by isosurfaces, the results are identical to rendered metaball surfaces presented by Blinn [1982]. These metaballs can be rendered by ray casting or by extracting a geometric representation. Marching Cubes [Lorensen and Cline, 1987] is the most prominent algorithm to extract the geometry of such a surface. However, sampling of the metaballs with too few points results in holes and visual artifacts.

[6] **Parts of this section have been published in**: M. Falk, M. Klann, M. Reuss, and T. Ertl. 3D visualization of concentrations from stochastic agent-based signal transduction simulations. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2010)*, pages 1301–1304, 2010.
M. Falk, S. Grottel, and T. Ertl. Interactive image-space volume visualization for dynamic particle simulations. In *Annual SIGRAD Conference*, pages 35–43, 2010.



▲ **Figure 5.21 —** Algorithm for computing a continuous density volume from particles. The 3D texture is updated slice-by-slice with the density contributions of the individual particles. The resulting volume is rendered with volume ray casting.

5.5.1 Technique

A 3D texture is used to store the density values. Evaluating Equation 5.7 for each voxel can be considered as gathering, since the contribution of all neighboring atoms is accumulated. The density volume can efficiently be computed on the GPU employing a scattering approach instead of the gathering approach [Kolb and Cuntz, 2005]. A similar GPU-based approach, however with a adaptively varying kernel support, was described by Cha et al. [2009] for the particle-based rendering of clouds. With scattering, each particle contributes only to the voxels it covers and the ensuing expensive neighborhood search is avoided. The contribution of a single particle is obtained by drawing its footprint at the position of the particle. The footprint is scaled so that it covers the support of the used filter kernel.

In Figure 5.21, the algorithm for computing the density texture is depicted. The density texture is updated slice by slice. Each slice has a thickness of one voxel and the density contribution of all particles has to be computed for all voxels of this slice. Every particle is splatted onto the current slice and the kernel function is evaluated for each covered voxel, taking the particle position and the voxel location into account. Particles which are too far away from the current slice can be discarded as they do not contribute to the density. The contributions of the individual particles are accumulated by using an additive blending operator. GPU-based volume ray casting [Stegmaier et al., 2005] is used for the visualization of the density including isosurfaces.

Empty Space Skipping

Assuming a small filter kernel support h and a mapping of density values of zero to full transparency, large parts of the volume will be empty. If the empty parts lie between the camera and the first non-empty voxel of the volume, the ray casting can be sped up by means of *empty space skipping* [Li et al., 2003]. Due to the construction with a radial-symmetric kernel function the extent of the filled volume is known. The maximum possible area that carries the density contribution of a particle is limited by a sphere around the particle with radius h. This information can be reused for empty space skipping.

The footprints of all particles are rendered into two textures. In the first pass, the footprints are moved toward the camera by h resulting in new start positions for the volume ray casting. By moving the footprint backward in a second pass, the end positions of the individual rays are obtained. Thus, only the parts of the volume between the updated start and end positions have to be sampled along the viewing rays. The possible speedup depends, however, on the number of particles, which have to be rendered to obtain the new positions, and, of course, the kernel support h.

5.5.2 Results

The time required for the computation of the density volume on the GPU was measured for a varying number of particles. The benchmark data model (cf. Appendix A.3) with a constant scene size is used as data source. All particles are splatted into the density volume, which encloses the whole cell with a diameter of 4 µm. The volume resolutions of 64 × 64 × 64, 128 × 128 × 128, and 256 × 256 × 256 thus correspond to voxel sizes of 62.6 nm, 31.2 nm, and 15.6 nm, respectively. For the splatting of the particles, three different splat sizes were chosen with diameters of 0.2 µm, 0.4 µm, and 0.8 µm. Table 5.7 shows the results in correlation with the number of rendered particles. The splat size affects the performance negatively because the number of covered voxels increases to the third power of the splat diameter. This also applies to the different volume resolutions. Doubling the resolution in all dimensions results in eight times the voxels covered for each particle. The extreme value of 301.66 ms for 500 000 particles might indicate that the GPU reached the maximum fill rate.

The performance of the GPU-based ray casting for volume rendering was measured with the simplified MAPK scenario (Section 4.5.1) at a viewport resolution of 1600 × 1200 pixels. The step size of the ray casting was set to a fourth the voxel size for a resolution of 128 × 128 × 128 and to a half voxel at 256 × 256 × 256. In Table 5.8, the results for volume rendering and volume rendering combined with isosurfaces are shown for four distinct camera settings. In addition, the effect of empty space skipping on the rendering times is depicted.

▼ **Table 5.7** — Measurements of the computation of continuous density volumes for different volume resolutions in the scaling scenario. Timings per frame are given in milliseconds [ms]. The hardware configuration is given in Appendix B.

Particles	64 × 64 × 64			128 × 128 × 128			256 × 256 × 256		
Splat Diam. [µm]	0.2	0.4	0.8	0.2	0.4	0.8	0.2	0.4	0.8
10 000	3.31	3.72	3.97	4.03	4.41	5.13	5.5	5.93	10.62
50 000	4.96	6.37	8.35	6.59	7.95	12.86	9.83	12.56	37.82
100 000	8.14	9.92	13.66	9.97	12.9	22.5	15.83	21.36	71.17
500 000	11.89	16.5	17.53	29.85	30.34	58.19	57.41	70.18	301.66

If empty space skipping is employed, the performance gain for volume rendering lies between 26 % and 59 % compared to plain volume rendering. This effect is only apparent when the camera is positioned outside the volume and the start positions of the ray can be adjusted. Inside the volume, all rays have to start at the near clipping plane right in front of the camera. In addition, the rendering of the particles results in very large glyphs due to the small distance toward the camera and hence the fill rate drastically increases. This overhead leads to an increase in the total rendering times by a factor of 6 for the smaller volume and a factor of 10 for the bigger one. The rendering of isosurfaces increases the time of the volume visualization by about 10 % except for the close-up setting. Here, the camera is located inside an area of high particle density and early ray termination sets in very early. Isosurfaces are hardly visible.

5.5.3 Application

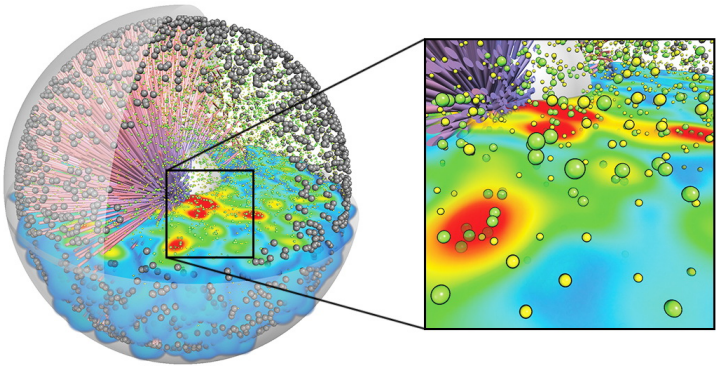
The continuous density representation can be applied to all kinds of particle data. In the following, two examples in the field of biology are presented as representative applications of this technique. The first example employs the continuous density to visualize the propagation of signaling molecules. In the second one, isosurfaces of the density volume are used to quickly approximate the surfaces of proteins, which are otherwise cumbersome to calculate.

Signaling Front in Cellular Systems

The relatively small, individual molecules are difficult to see in the context of the comparatively large and crowded cell. Although every molecule is equally important for the signal transduction process, the knowledge of just one molecule position is not sufficient to understand the signaling outcome. A continuous representation is in contrast easier to interpret and more comparable to the results of non-spatial simulations. However, the spatial context is missing if only the volumetric density data is shown. By combining the visualization of the density volume with the schematic visualization, the spatial context

▼ **Table 5.8 —** Measurements of the volume rendering of the MAPK scenario. Timings per frame are given in milliseconds [ms] and were measured for DVR (vol), isosurfaces combined with DVR (iso), and DVR with empty-space skipping (space) at volume resolutions of 128 × 128 × 128 and 256 × 256 × 256. The hardware configuration is given in Appendix B.

Setting	128 × 128 × 128			256 × 256 × 256		
	vol	iso	space	vol	iso	space
Viewport covered	153.492	169.319	110.852	265.393	296.824	167.14
Full height	141.583	158.153	108.342	259.808	291.121	195.274
Half height	67.668	74.444	53.565	128.783	140.509	101.937
Closeup	3.576	3.672	23.43	3.465	3.475	35.1



▲ **Figure 5.22** — Combined schematic visualization and continuous signal concentration of MAPK proteins. The visualization includes membrane-bound receptors (gray), phosphatases (yellow), and MAPK signaling proteins (green). Low concentrations are mapped to blue, high to red.

is given by the geometrical objects and the remaining space is filled with the results of the volume rendering. Figure 5.22 shows a cell model, where the signal distribution is shown by means of the continuous density volume.[7]

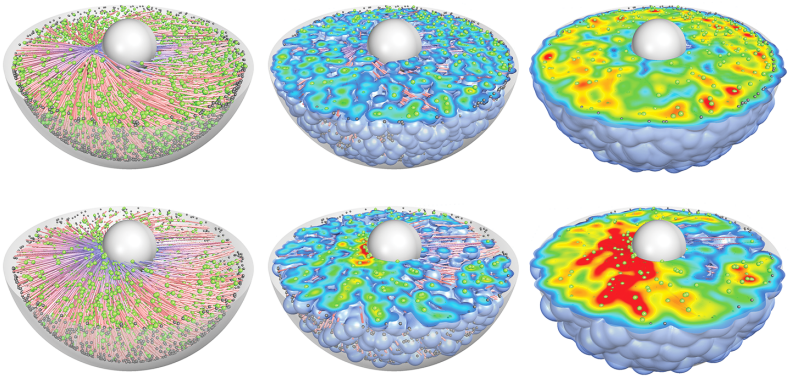
Accordingly, the continuous density distributions of the signaling particles can be used to visualize local, sub-cellular properties. Tuning the respective local filter kernel leads to a transition from discrete particles via discontinuous local accumulations of molecules towards a global concentration distribution in the cell, showing how the signal is in fact transported (see Figure 5.23). The global distribution of the individual molecules reveals the effect of the motorized transport when compared to the signal transport by sole diffusion. While the particles are distributed only near the plasma membrane due to diffusion (Figure 5.23, top), motor proteins transport particles far into the cytoplasm (Figure 5.23, bottom).

Since the density concentration is computable in a few milliseconds, following the temporal development of the signal is possible as well as the interactive in-depth exploration of signal transduction properties. In Figure 5.24, the propagation of MAPK molecules is shown for a thin transection of a cell. Due to the polarized cytoskeleton and the motorized transport, the signaling molecules accumulate on the left-hand side of the nucleus resulting in high signal concentrations.

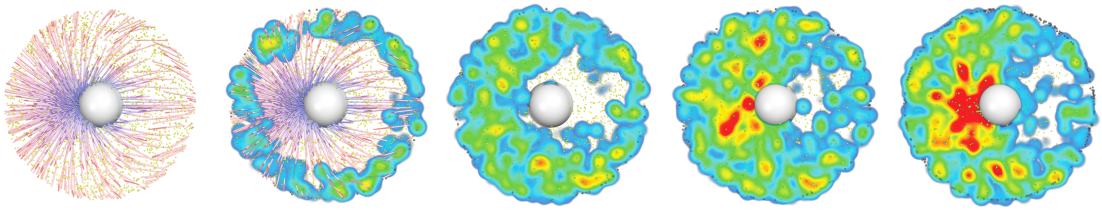
Interactive Exploration Protein Cavities

In the presence of substrate, many proteins open up substrate channels toward their active site. For domain experts it is of high importance to locate such substrate channels and track them over time. One promising approach is to locate cavities near the active site and

[7] **Parts of this section have been published in**: M. Falk, M. Klann, M. Reuss, and T. Ertl. 3D visualization of concentrations from stochastic agent-based signal transduction simulations. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2010)*, pages 1301–1304, 2010.



▶ **Figure 5.23** — Comparison of the signal transportation by diffusion (top) and motorized transport (bottom). Adjusting the kernel support (left to right) allows the transition from particles to a continuum.



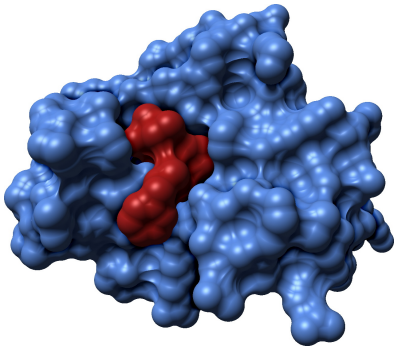
(a) $t = 0\,\mathrm{s}$ (b) $t = 4\,\mathrm{s}$ (c) $t = 8\,\mathrm{s}$ (d) $t = 12\,\mathrm{s}$ (e) $t = 50\,\mathrm{s}$

▲ **Figure 5.24** — Time series of the signal propagation of a MAPK simulation obtained by visualizing a continuous particle distribution.

to track them until they reach the surface. Available tools like PocketPicker [Weisel et al., 2007] only work with a single, static snapshot and are, hence, not suitable for studying time-dependent data resulting from MD simulations. However, using a fast volume segmentation in combination with the interactive construction of a density volume of the atoms enables the interactive investigation of the evolution of protein cavities over time.[8]

The first step is to extract the boundary of the molecule, i.e. the molecular surface, with respect to the surrounding substrate. The solvent excluded surface (SES) [Richards, 1977] represents the molecular surface that is accessible by solvent molecules (see Figure 5.25). It is obtained by rolling the solvent molecule, i.e. the probe, over the van der Waals surfaces of the individual constituting atoms of the molecule. Since the computation of this surface is computationally challenging and consumes too much time, a density volume is used instead for the approximation of the SES. The continuous density volume is generated from the atom positions of the input protein and their respective radii. The SES is extracted from the density volume by means of an isosurface. By adjusting the support of the filter kernel and the isovalue, different probe radii can be realized. The approximation error is barely noticeable and can be neglected in the follow-up steps because an opening between two cavities is visible if and only if the solvent molecule can pass through. The density volume, which is generated on the GPU, is transferred into the

[8] **Parts of this section have been published in**: M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl. Interactive exploration of protein cavities. *Computer Graphics Forum*, 30(3):673–682, 2011.



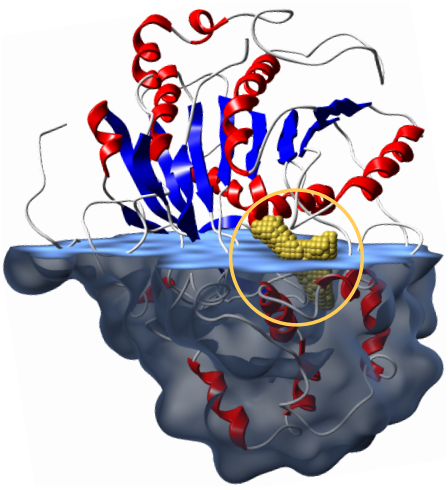
◀ **Figure 5.25 —** Solvent excluded surface (SES) of a small protein (isomerase FKBP12, blue) with a ligand (red). The ligand is docked to the protein, i.e. it is not intersecting the protein surface but rather fits like a key to the lock.

main memory. Partial volume segmentation is applied to a user-selected cavity and the cavity is tracked over several time steps.

The approach was integrated by Michael Krone into a standalone prototype, which facilitates a user-driven, interactive visual analysis of protein cavities [Krone et al., 2011]. The tracking of a cavity is initiated by the user through selecting a cavity in a cross-section view of the protein. The cavity is automatically tracked over time until its size changes dramatically. At this point, the cavity might have reached the surface and a substrate channel is formed or two cavities were merged. The user can then continue the tracking of the same cavity or select a different one. The size of the cavity is plotted over time in an additional graph. At a volume resolution of $256 \times 256 \times 256$, the approach performed with 3 fps and 8 fps for proteins consisting of up to 60 000 atoms (see [Krone et al., 2011] for details). The applicability of the approach is demonstrated by a small user study with two data sets, namely a lipase A from *Candida antarctica* (PDB ID: 2EVO) and the *Rhizomucor miehei* lipase (PDB ID: 3TGL) to investigate the structural influences of organic solvents. Figure 5.26 shows the results of a cavity segmentation in an enzyme. The filled cavity is visualized by yellow spheres for each segmented voxel.

5.6 Image-space Volume Visualization of Particle Data

The technique described in Section 5.5 uses a uniform 3D grid in object space to store the densities of particles. The resolution of volume is, however, limited due to high memory requirements. A volume storing one scalar value as a single precision floating point value requires 512 MiB for 512^3 voxels and hence 4 GiB for 1024^3 voxels. In addition, the time required for generating the volume also increases and is confined by the maximum available fill rate of the GPU. Because of the volume resolution, the minimum size of visible details, which are enclosed in the volumetric data, is also limited. This specific problem can partly be solved by attaching the volume to the front of the camera. That is, if the camera is positioned somewhere inside the data, the density volume would only be recomputed for the visible particles in front of the camera and not for the whole data

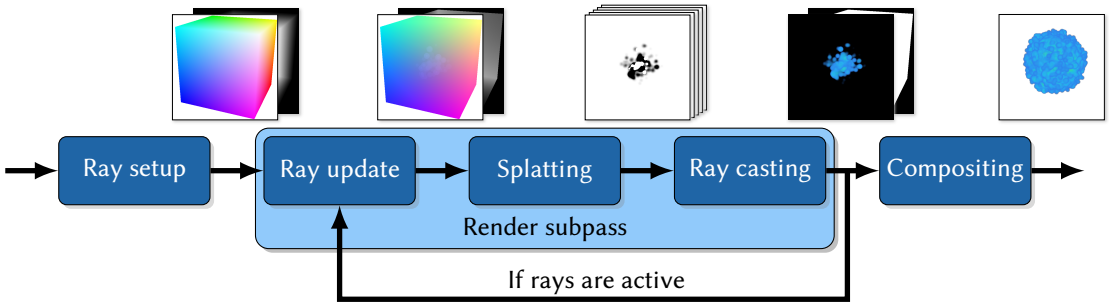


▶ **Figure 5.26 —** Segmentation of a protein cavity employing a continuous density volume of the atoms of the protein. The selected cavity (yellow circle) inside the enzyme isomerase is visualized by yellow spheres for each segmented voxel. On the lower half, the isosurface of the protein's SES approximation is shown.

set. Thus, the volume resolution remains constant independent of camera movement and details become visible when the particles are close to the camera. Fraedrich et al. [2010] extend this basic idea by using a volume, which is adjusted to the perspective projection of the camera to fill the entire image space. The result is a volume with equally sized voxels in image space and frusta of different sizes in object space, respectively. The resolution limit of the volumetric data, however, remains and the sampling of the volumetric data is more complicated because of the changing between the projection space of the camera and object space. Two iterative approaches of ray casting metaballs, i.e. the isosurfaces of particles, were proposed by Müller et al. [2007]. However, both methods only achieve interactivity for medium-sized data sets and, in addition, are only capable of rendering the metaball surfaces while neglecting volume rendering. In the following, a technique is presented that bypasses the resolution limit of volumetric textures by generating and visualizing a virtual volume at the current viewport resolution on the fly with both isosurfaces and DVR.[9]

In this approach, the general ideas of texture slicing, ray casting, and splatting are combined in a novel way. The density volume resulting from the particles is reconstructed in screen space at viewport resolution. The reconstruction is performed only for those positions that are to be sampled in the subsequent volume rendering step. The issue of perspective correction is, thus, moved from volume rendering to data reconstruction. Volume data reconstruction and the sampling are tightly interleaved in a GPU-based rendering pipeline. Employing a combination of slice-based ray casting and a dynamically evaluated density volume, the required memory footprint can be reduced dramatically in comparison to uniform grids. In addition, fine details become visible because of the high volume resolution, which is identical to the resolution of the viewport. The overall performance is optimized by employing inter-frame occlusion queries to determine the termination of the ray casting.

[9] **Parts of this section have been published in:** M. Falk, S. Grottel, and T. Ertl. Interactive image-space volume visualization for dynamic particle simulations. In *Annual SIGRAD Conference*, pages 35–43, 2010.

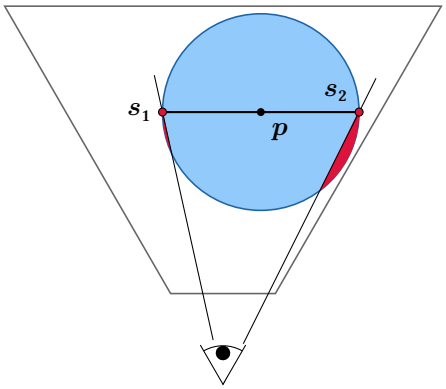


▲ **Figure 5.27** — Algorithm for computing and visualizing a continuous density volume in image space. Ray casting and splatting are combined to generate and sample the density contribution of particles on the fly in several rendering passes.

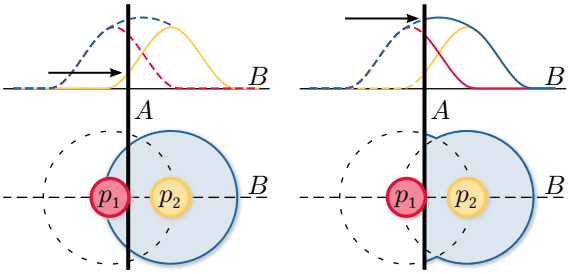
5.6.1 Technique

Figure 5.27 shows the algorithm of the image-space approach on the GPU. Each step in the depicted algorithm is performed in a separate shader. First, the initial rays needed for the ray casting have to be set up. The sampling along the rays is carried out on concentric shells around the camera. The first shell touches the point of the data's bounding box that is closest to the camera to skip the non-occupied, empty space. The distance between two adjacent shells is given by the volume sampling distance $\Delta\lambda$. The volume rendering and, hence, the volume reconstruction is split into several rendering passes until all rays have been terminated.

In each subpass, the density volume is reconstructed for the current spherical shell by splatting all particles onto the image plane. Before the splatting takes place, the ray positions are updated to lie on the current shell. The particles, which do not contribute to the density values at the sampling positions on the spherical shell, are discarded in the vertex shader. Sorting the particles along the main viewing axis can be used to reduce the workload of the GPU because less particles have to processed by the vertex shader for a single shell. For the splatting itself, the particle footprints are drawn and the kernel function (cf. Equation 5.7) is evaluated for each covered pixel at the respective sampling position of the ray. However, if only the circular particle footprint is projected from object space into image space, parts of the 3D radial-symmetric kernel function might not be accounted for. Figure 5.28 illustrates this problem for a single particle. Since only the space between s_1 and s_2 of the 2D footprint is considered, two spherical caps are left out of the contribution of the particle. The splats of the kernel function, therefore, have to be perspectively corrected to avoid this problem. This is achieved by handling the particles as spherical glyphs during vertex processing as described in Section 2.5.2. To reduce the number of draw calls for the particle splats, the density values of different depths, i.e. multiple shells, can be evaluated in a single pass by employing multiple render targets. After splatting, the accumulated densities stored on the shells are sampled in the next step by performing volume ray casting along the viewing rays. During the ray casting step, the depth buffer is modified for all rays that are terminated due to early ray termination or leaving the bounding



▶ **Figure 5.28** — Point splats have to be perspectively corrected when projected from object space into image space. If, for perspective projection, only the splat footprint between s_1 and s_2 is considered, parts of the volumetric splat (red) are missing in image space.



▶ **Figure 5.29** — Correct volumetric density clipping. If particle p_1 is clipped, its density contribution is neglected leading to a wrong density (left). For correct volumetric clipping, p_1 has to be considered (right). A denotes the clipping plane, the density profiles are shown for the cross-sections B.

box of the data. After the ray casting, the number of active rays is obtained with hardware-based occlusion queries. These queries count the fragments that pass the depth test. Thus, it is possible to count the number of active rays, if the depth buffer is prepared according to already terminated rays. The occlusion queries are performed during ray update, but the results of the queries are not immediately available. To hide the latency, frame-to-frame coherence is exploited and the results are queried in subsequent frames. The number of remaining, active rays is then used to adjust the number of necessary render passes. In the final step, the results of the ray casting are composited into the displayed framebuffer.

The hardware clipping planes of OpenGL can be used to cut away parts of the volume. However, when particles are clipped at the intended cut planes, the final volume will show no sharp cuts because the kernel is evaluated for the complete particle footprints (Figure 5.29, left). Additionally, the missing contribution of clipped particles leads to wrong results. Thus, the clipping is performed in the fragment program during density computation. The hardware clipping planes are shifted outward so that only non-contributing particles are discarded. The density computed in (5.7) is set to zero if the sampling position $\boldsymbol{p}$ is outside the original, non-shifted clipping planes. The final result is a correct volumetric clipping as illustrated in Figure 5.29, right.

5.6.2 Results

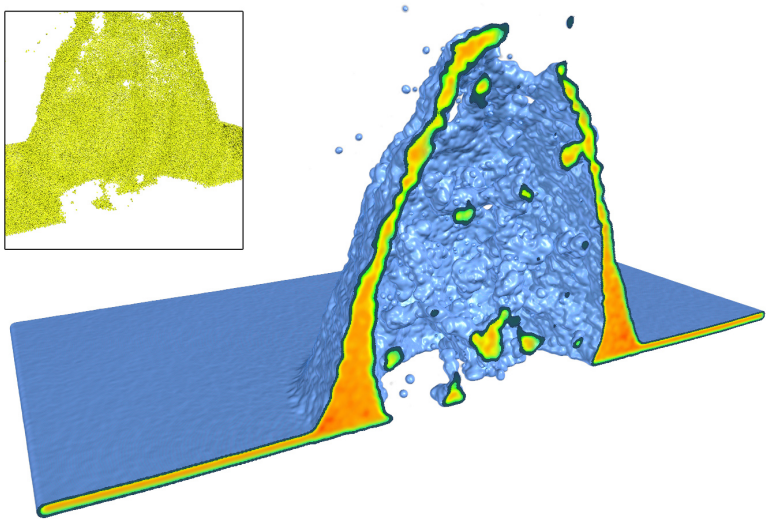
The performance of the image-space approach was measured for different data sets. In Table 5.9, the results are shown and compared with measurements of the object-space approach using a uniform grid (cf. Section 5.5).

A viewport resolution of 1600 × 1200 was used throughout the measurements and the resolution of the uniform grid was set to 256 × 256 × 256 voxels. The uniform grid was chosen to enclose the complete data set. Depending on the characteristics of the data set, the time required by the image-space approach is about 4 to 10 times higher than the computation and rendering in object space. However, the rendering resolution is much higher since it matches the viewport resolution. The isosurfaces were rendered semitransparent without additional volume rendering. The isosurface rendering, hence, requires more time since early ray termination does not take place and many rays have to traverse the whole data set.

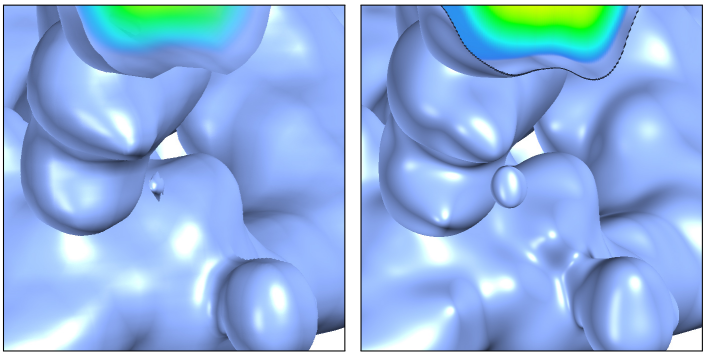
The data set shown in Figure 5.30 results from an MD simulation that was coupled with finite element analysis. A pulsed laser beam heats up a solid metal block and a bulge is formed. The cutaway reveals the density distribution inside the material and contains roughly 250 000 particles. Glyph-based rendering of the particles (Figure 5.30, top left) hardly reveals any structures in static images. In Figure 5.31, a closeup of Figure 5.30 is depicted that has been rendered with the object-space and the image-space approach. Because of the finite resolution of the uniform grid, fine details are missing as illustrated in Figure 5.31(a) and sharp corners get smoothed out due trilinear interpolation. The geometric features are conveyed using the image-space approach (Figure 5.31(b)). Small features like the ellipsoid in the center of the closeup are fully captured with the image space method, whereas the features are deformed and barely visible in object space.

▼ **Table 5.9 —** Measurements of the image-space volume visualization of different particle data sets with both volumetric and isosurface rendering. Timings per frame are given in seconds [s]. The hardware configuration is given in Appendix B.

Structure	# Particles	Image Space		Object Space	
		Volume	Iso	Volume	Iso
Splat (laser ablation)	892 219	0.625	1.075	0.159	0.155
Bulge (laser ablation)	509 423	0.84	2.174	0.123	0.121
Cell	103 284	0.25	0.282	0.042	0.06
Viral envelope (1SVA)	958 980	2.564	5.882	0.267	0.392
Ribosome (2WD*)	147 236	1.333	2.222	0.144	0.189
Insulin (1RWE)	823	0.216	0.372	0.048	0.057



▲ **Figure 5.30** — Laser ablation leads to a bulge on a metal block. Particle rendering of a cutaway of the bulge with glyphs is not sufficient to grasp the overall shape in still images (top left). Visualizing the same data with the image-space approach reveals the shape and the density distribution inside. Laser ablation data set courtesy of Steffen Sonntag.



(a) (b)

▶ **Figure 5.31** — Closeups of a small region at the bottom of Figure 5.30. (a) object-space density volume. (b) image-space density volume. Ridges, grooves, highlights, and small features are more distinct in image space.

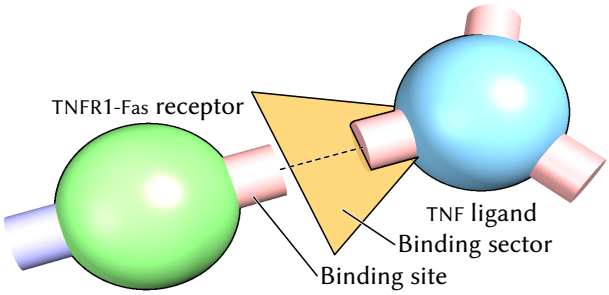
5.7 Visualization of Membrane-bound Receptor Clustering

In Section 2.1.2, the extrinsic pro-apoptotic pathway and its activation is described. Ligand-receptor clusters that are located at the plasma membrane are the key players in the activation process of procaspase 8. It is, hence, of high relevance to understand the dynamics of such ligand-receptor clusters. This includes the building process, the stability, and the temporal developments of these clusters. Markus Daub developed a mathematical, *in silico* model to implement the clustering with a stochastic simulation for a small region of the plasma membrane. The basic principles of the schematic visualization approach (cf. Section 5.2) are slightly adjusted to incorporate all data given by the mathematical model. The focus of the visualization lies on the internal structure of clusters instead of approximating these clusters, e.g., with ellipsoids [Grottel et al., 2007]. The visualization of the simulation results serves as a tool for visual inspection of the data. Additionally, it can be used to emphasize irregularities like invalid bounds hidden in the data, which is useful for detecting bugs in the simulation code. Thus, the visual analysis can be seen as an integral part of the development cycle of the model on receptor clustering.[10]

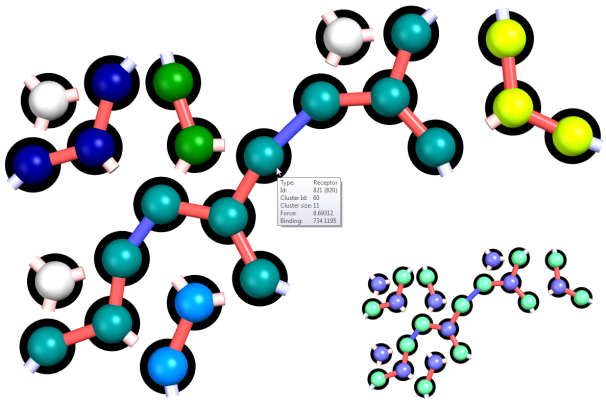
The simulation domain of the *in silico* model comprises only a part of the cellular plasma membrane. A region of $1\,\mu m \times 1\,\mu m$ is selected and flattened, thereby restricting the stochastic simulation to two dimensions. To mimic the closed shape of the plasma membrane, periodic boundary conditions are used. In the model, ligands and receptors are both represented by particles with a predefined radius. The movement of both particle types is restricted to the flattened section of the membrane. The diffusion process of the particles is modeled by a Langevin equation, i.e. a SDE, describing a Brownian motion. Lennard-Jones potentials are used to model attraction and repulsion of the individual ligands and receptors. Since the particle movements are constrained to a plane, it is sufficient to include only the particle rotation around an axis, which is perpendicular to the plane. In the *in vivo* environment of cells, this rotation constraint is also given for the membrane-bound receptors. The coupling of the diffusion with molecule interactions and molecule rotations yields a system of non-linearly coupled SDEs. This system of SDEs is solved with a particle simulation [see Falk et al., 2011a].

The receptor (TNFR1-Fas) is modeled as a sphere with two docking sites, one for another receptor and the second one for binding to a ligand (see Figure 5.32). The TNF ligand has three binding sites and is also modeled by a sphere. The binding sites are explicitly modeled and considered throughout the simulation with respect to the particles rotation. Bonds between ligands and receptors emerge when the binding sites of both feature the correct orientation, i.e. opposed directions with a small deviation. In the simulation, bonds are possible if the direct connection between the binding sites does not leave a angular sector around the binding sites. Once a bond has been established, it can break

[10] **Parts of this section have been published in**: M. Falk, M. Daub, G. Schneider, and T. Ertl. Modeling and visualization of receptor clustering on the cellular membrane. In *IEEE Symposium on Biological Data Visualization (BioVis 2011)*, pages 9–15, 2011.



▶ **Figure 5.32** — Glyph-based representations of a TNFR1-Fas receptor and a TNF ligand.

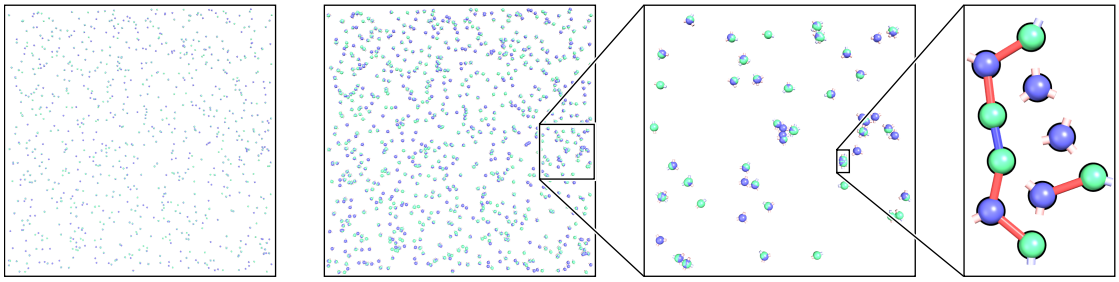


▶ **Figure 5.33** — Clustering of receptors and ligands on the plasma membrane. The conformational information of the clusters, i.e. the location of ligands and receptors, is shown on the lower right. Tool tips provide additional information.

if the binding sites between ligand and receptor are no longer correctly oriented due to molecule movement or if the molecules are too far apart. For the visualization of the simulation results, glyph-based rendering is employed.

5.7.1 Technique

The data resulting from the simulation is processed prior to the visualization. In this preprocessing step, local clusters, i.e. connected components, consisting of ligands and receptors are extracted. A depth-first search is employed to assign the same cluster ID to all molecules of an interconnected cluster. In addition, the chemical bounds representing the connectivity between molecules are stored. Both ligands and receptors are represented by spherical glyphs. The binding sites of the individual molecules are rendered as cylindrical glyphs. Additionally, different colors are used for the binding sites to indicate the type of the potential bond, i.e. either a ligand-receptor bond or a receptor-receptor bond. The bonds themselves are visualized by cylinders while the respective binding sites are hidden. The bond connects both molecule centers and might therefore deviate from the direction of the binding site. That is because the molecules may not be properly aligned due to the usage of the binding sectors. Bonds between molecules feature highly saturated colors whereas unbound binding sites are displayed with less saturation. Figure 5.33 shows eight separate clusters and the respective bonds, where an apex angle of 120° was used for the binding sites in the simulation.



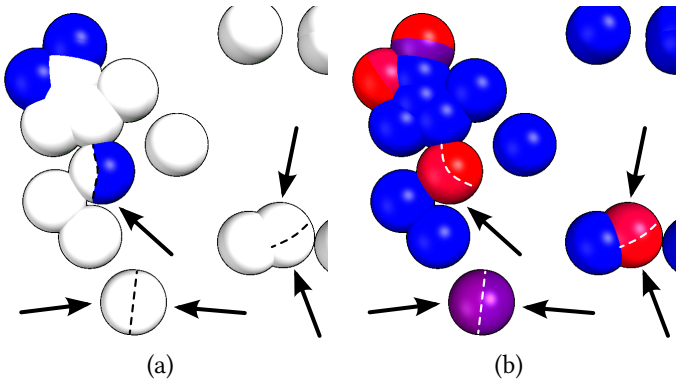
▲ **Figure 5.34 —** Dynamic, depth-dependent scaling for ligand-receptor clusters. (a) simulation domain with unscaled radii. (b) radii scaling of 2. (c) unscaled radii. (d) downscaled radii to enhance bounds (scaling factor 0.05).

Different coloring modes for the molecules convey either local information, like molecular species and acting force, or more global information, like, e.g., cluster size or cluster ID (cf. Figure 5.33). The rendering of the spherical molecules and the cylindrical binding sites is separated into separate render passes to provide more flexibility. Since the simulation domain and, hence, the data is two-dimensional, the third dimension can be used to offset individual parts of the visualization. Ligands, receptors, binding sites, bonds, and trajectories are all stored in different layers and can be stacked on top of each other by a user-given offset. The visibility of the individual layers can be toggled allowing the user to adjust the visualization to his needs. The layering approach is most useful for inspecting bonds or trajectories of single molecules, since in both cases the information is often hidden by the molecules themselves. Unique molecule IDs, which do not change during the simulation, allow for single molecule tracking. The trajectories of molecules are also rendered with GPU-based glyphs.

The differences in scale that are inherent to the model pose a problem for the visualization. The scales range from the data domain (µm) over molecules (nm) down to their movements (pm). When the whole data set is shown, the individual, unscaled molecules are too small to be distinguishable (Figure 5.34(a)). If, however, distance-dependent scaling is enabled with a scaling factor of 0.05 to 2, exploration is possible without further adjustments as illustrated in Figure 5.34(b), (c), and (d). Only the radii of molecules and their respective binding sites are affected by the scaling, but not their positions. The scaling factor is independently computed for each molecule based on the squared distance between the position of the molecule and the camera. The range of scaling was chosen so that individual molecules remain discernible at every scale and that bounds become visible when visualizing single clusters.

5.7.2 Qualitative Results

The model consisting of 1000 ligands and 1000 receptors was simulated for approximately 500 million time steps with Δt set to 10^{-9} s. This corresponds to a total simulation time of 0.5 s. All molecules were initially positioned randomly inside the simulation domain



▶ **Figure 5.35 —** Using colors to reveal inconsistencies in cluster data. (a) colors matching cluster size (white: cluster size of one, blue: size of two). (b) force coloring (blue: low force, red: high force).

without any existent bonds. For the subsequent visualization only every 100 000th time step was stored to disk. About 28 days were required to perform the simulation with a non-parallel implementation on the hardware configuration given in Appendix B.

The visualization of the data itself is highly interactive. On the same hardware, the glyph-based rendering of the molecules and their binding sites exceeds 1000 fps. The different color schemes have distinct applications in the visualization. Apart from the distribution of receptors and ligands, they reveal important aspects of the data that otherwise would be difficult to spot. In Figure 5.35, an example is depicted where toggling the coloring mode reveals some inconsistencies in the simulated data. The particles indicated by the arrows feature a high acting force, which implies interactions, i.e. the building of a cluster. However, there exist no clusters with two or more molecules in the vicinity.

CHAPTER 6

The CellVis Framework

In the course of this thesis, the individual building blocks for model development were designed, implemented as prototypes, and finally assembled into a single framework following the development cycle of biological models (cf. Figure 1.2). These building blocks can be used independently, but then the user has to switch between different applications. Thus, the need arises for a unified system that incorporates the separate steps or at least provides a common interface between them. The framework named *CellVis* unites modeling (Chapter 3), simulation (Chapter 4), and visualization (Chapter 5). The overall design process was driven by a computing perspective. The aim was to design a user-friendly system that should be usable by biology experts who do not necessarily have a strong background in computer science. During the development of CellVis, a close collaboration with biologists ensured that their needs and expectations were considered.

The CellVis framework is not the first of its kind. There exists a wide variety of systems that are concerned with the modeling and simulation of cellular systems. However, the main focus of CellVis lies in the utilization of advanced visualization techniques for the analysis of the simulation results. CellDesigner [Funahashi et al., 2008], TinkerCell [Chandran et al., 2009], and BioNetCAD [Rialle et al., 2010] are just a few examples of integrated systems. A recent overview of existing, spatial simulation packages is given by Klann and Koeppl [2012]. For the simulation of the models, ODE solvers are most commonly used. Only a few simulation packages like BioNetCAD rely on particle-based simulations. Performing the simulations in lock-step with the analysis is not considered within these packages. This might be due to short simulation times of ODE solvers or the need and possibility simply did not arise. Casanova et al. [2004], however, propose a "*scientific virtual instrument*" allowing the user to interactively steer simulations with MCell [Stiles and Bartol, 2001] on compute clusters. Their approach is an example of *computational steering* (cf. Section 2.5) and the simulation can be adjusted on-the-fly to focus on regions of interest. Since the simulation, which is proposed in this thesis, is

not grid-based like in MCell, selecting regions of interest hardly makes sense during the simulation. That is because the particles located outside the region of interest still have to be computed. If this is not the case, the molecule distributions will no longer be consistent. On the other hand, computational steering can be used in particle-based simulations to adjust various parameters, like reaction rate constants, molecule radii, or even the number of molecules, between two time steps. In CellVis, this is possible by a tight coupling of both, simulation and visualization. In the following, the software architecture of the framework is briefly described and the applicability of CellVis is shown for different use cases.

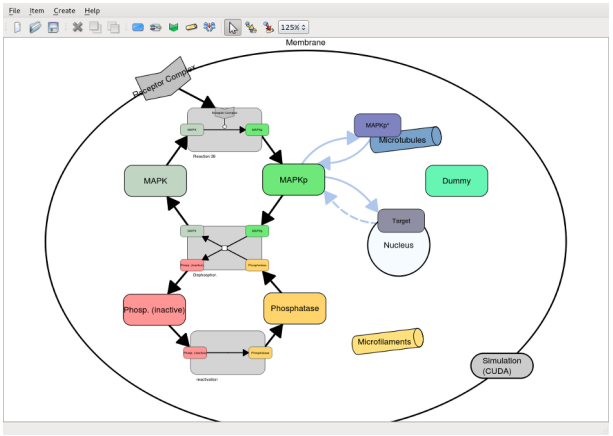
6.1 Architecture of CellVis

As mentioned before, the process of model development includes the subsequent steps of model design, simulation, and analysis. The CellVis system provides a unified interface to combine these three steps into one interactive tool. When the components are used independently, the data exchange is handled by a common data format. A client-server model provides a common interface to enable computational steering of the simulation in lock-step with the visualization.

6.1.1 Modeler

The modeling prototype employs the graphical notation described in Section 3.2. The process of assembling a cellular model is done in a graph-based manner and the model configuration itself is stored in an XML format. Every node in the graph has a graphical representation and a data representation. The graphical representation defines the appearance and position inside the model. In addition, visualization parameters of the analysis step can be saved in the graphical representation. All parameters and settings concerned with the simulation are handled by the data part of the nodes.

The modeler can be used either as stand-alone application or tightly integrated into the visualization component. The stand-alone usage might be preferred to prototype a new model or adjust an existing model configuration prior to simulation. Figure 6.1 shows a screenshot of the prototype. The plasma membrane is automatically fitted to enclose all cellular components. Reactions and interactions are created by dragging molecules onto the reaction container or the cellular structure, respectively. The reaction type is determined depending on the number of connected reactants and products. In the visualization prototype, the modeler is embedded as a separate window to handle the import of the configuration from the XML file as well as adjustments of molecule definitions.



▶ **Figure 6.1** — Screenshot of the modeling prototype showing the cellular model of the simplified MAPK pathway.

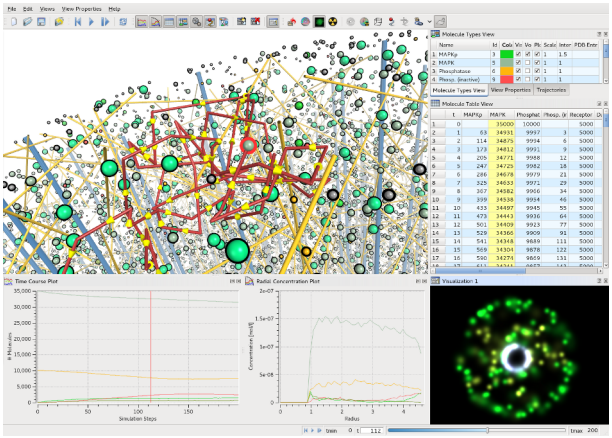
6.1.2 Simulation

The simulation accepts the XML configuration from the modeler and extracts the respective parameter set. After building the cellular model according to the parameters, the simulation takes place either on the CPU or on the GPU. The simulation front end is a command line application that also acts as server. If visualization clients are registered at the simulation server, the data of single time steps is sent to clients when the computation of a time step is finished. In the stand-alone mode, the data results are stored in permanent storage for later analysis.

6.1.3 Visualization

Data analysis is an important step in the process of model development. Since the focus of this thesis lies primarily on the interactive visualization of biological systems, the main part of the visualization prototype consists of a 3D visualization view. The primarily used technique is the schematic visualization (cf. Section 5.2). This sparse representation can be enhanced by a continuous density distribution of the molecules (cf. Section 5.5). The microscopic visualization approach is also provided in the framework as secondary visualization (cf. Section 5.4). In the visualizations, clipping planes can be arbitrarily placed and the user can select the elements which are clipped by each plane. To provide different views on the data, multiple coordinated views [Roberts, 2007] can be enabled. Multiple 3D views can be linked together by using a shared camera, i.e. camera movements in either view affect all linked views. The user has full control over the camera to explore the data set.

In addition to the visual representations of the cell model, 2D plots and tables are provided. To understand the time-dependent behavior of the model, one table holds the data of all time steps and for each time step the molecule number of each molecular species is shown. The contents of this table are also represented graphically over time in the time course plot. A second graph shows the radial concentration of the current time step. To obtain the radial concentrations, the cell is subdivided into spherical shells and the number of



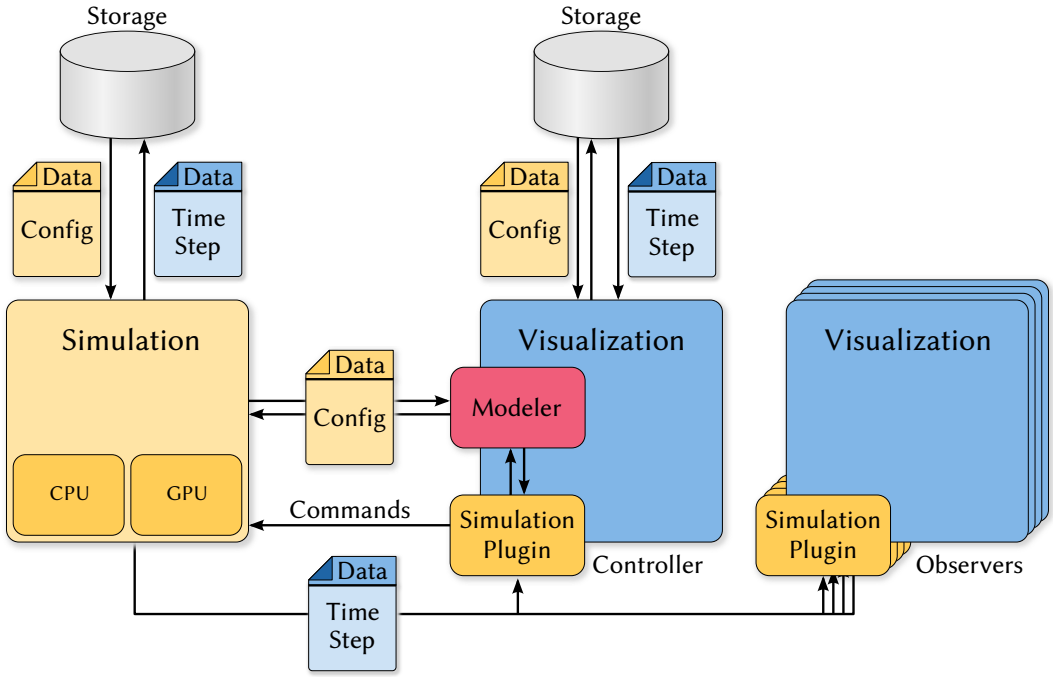
◀ **Figure 6.2 —** Screenshot of the visualization prototype showing the simulation results of the simplified MAPK pathway.

molecules per shell is determined. The volumetric concentration is obtained by the ratio between molecules per shell and the volume of the shell. For each molecule type, the resulting concentrations are plotted against the cell radius.

In Figure 6.2, a screenshot of a typical analysis session in CellVis is shown. The main view (top left) shows the simulation results of the simplified MAPK pathway with the schematic visualization. The trajectory of a single MAPK molecule is highlighted. The time series of the simulation, i.e. the development of molecule numbers during the simulation, and the radial concentration of the current time step are plotted in the lower part of the application. In addition to the time course plot, the molecule numbers are also presented in a tabular form (center right). An overview over the whole cell is given in a second visualization view by using the microscopic visualization (lower right). The data is handled in a second thread to be separated from the visualization. Several time steps are cached in advance for a smooth playback of the time series. The visualizations themselves are responsible for moving the particle data into GPU memory.

6.1.4 Integration of the Components

The overall architecture of the CellVis framework is shown in Figure 6.3. The visualization component is the centerpiece of the framework. It contains one instance of the modeling prototype for data handling. The simulation can be accessed via a plugin mechanism over network sockets. By using network communication, the simulation server is not restricted to be on the same computer where the analysis takes place, but can, e.g., be executed on a remote computer with dedicated hardware. Once the visualization client is registered at the simulation server, the client controls the current simulation. The model configuration might be modified in parts or as a whole. Additional clients may register at the server but only the client registered first can affect the simulation. If the simulation is to be performed in lock-step with the visualization, the visualization has to trigger the simulation. When the visualization requests the next time step, the simulation is resumed for a single step. After the simulation step is finished, all registered visualization clients are notified and the data is sent over the network.



▲ **Figure 6.3 —** Software architecture of the CellVis framework. The three steps of the model development cycle—modeling, simulation, and visualization—are tightly coupled and connected over network sockets.

By using the embedded model editor in the visualization, it is possible to adjust simulation parameters like diffusion coefficients or reaction rates. These changes are forwarded to the simulation and immediately affect the simulation. With such a coupling, effects of rate changes can be studied in an easy manner. Offline data analysis can be performed on simulation results computed beforehand. This offline analysis is a useful tool for larger data sets, both in particle numbers and simulated time steps.

6.2 Applications

All visualization techniques presented in Chapter 5 are integrated into the CellVis framework, with the exception of the image-space volume visualization. The overall performance of the prototype was not explicitly measured since the limiting factors are determined by the respective data sets and the applied visualizations. The overhead inflicted by the user interface is negligible.

CellVis is used for the visualization of the data resulting from the simulations of cellular models. The analysis of data from ligand-receptor clustering simulations is supported by additional, specialized views (cf. Section 5.7). In particular, a histogram for different cluster sizes is provided besides a plot showing the temporal development of clusters, thus enabling visual exploration by the user. The visualization component of CellVis,

despite primarily intended for the visualization of cellular, biological data sets, is generally applicable to visualize particle data from different sources. These data sets include, e.g., data obtained from laser ablation simulations (cf. Section 5.6) and, as depicted in the following, drug diffusion into a solid tumor and the results of a galaxy simulation. Both glyph-based and volumetric rendering approaches are employed to visualize the data sets.

6.2.1 Cellular Simulations

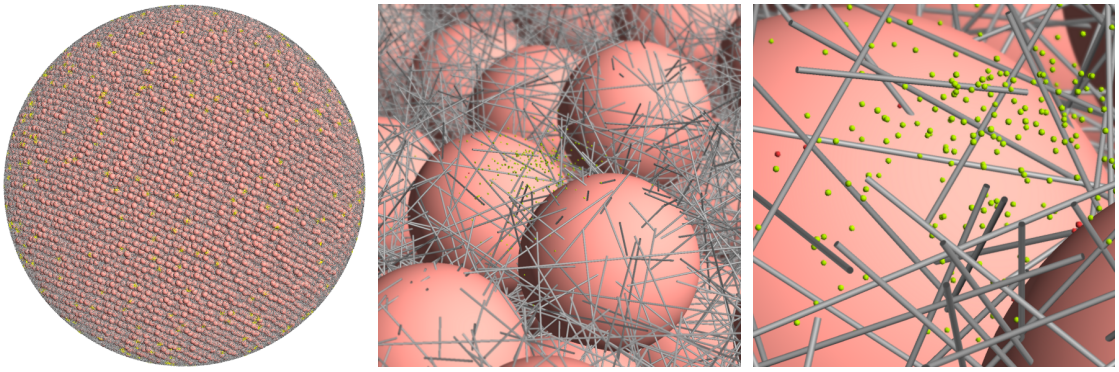
Several cellular models are introduced in this thesis and simulated with a particle-based simulation. In this work, all images illustrating the simulation results were created with CellVis. See Appendix A for the descriptions of the respective cellular models. Here, Figure 6.2 serves as a representative screenshot for the visualization of cellular data sets.

As mentioned before, the simulation and the visualization can either be carried out in lock-step mode or on their own. Small models, i.e. containing low molecular populations, or short time scales are perfectly suitable for interactive computational steering. If the simulation, however, requires minutes or hours to complete a single time step, employing computational steering is not sensible and an offline analysis is advisable. When used for the offline analysis of intermediate simulation results, CellVis is a valuable tool to obtain early feedback of the simulation. The whole framework can thus serve as test bed for parameter estimation. Parameters can be adjusted and a fast, exemplary simulation is performed by using large simulation time steps to evaluate the parameter settings. The simulation results are not necessarily precise due to the larger time steps, but the general idea of how the systems behaves might be confirmed.

6.2.2 Drug Diffusion into a Tumor

The simulation approach described in Chapter 4 can also be applied to models larger than a single cell. In the following application scenario, a solid, spherical tumor is modeled by interpreting spherical obstacles as individual cancerous cells and the cytoskeleton representing fibers of the extracellular matrix. The tumor is about 1 mm in size and at this size, nutrient supply without additional blood vessels is still possible by diffusion. The constituting cells are tightly packed and held in place by fibrous structures. Drug molecules are injected at a constant rate into the tumor at the boundary of the tumor. The drug is designed to trigger the death of the cancer cells [Dreher et al., 2006]. The molecules diffuse into the tumor and are hindered by cells and fibers. Once the drug molecules are in the vicinity of a cancerous cell they can adsorb to the surface of the cell. The particle-based simulation is well suited to cope with the different orders of magnitude between the tumor (1.2 mm) and the drug molecules (20 nm).[1]

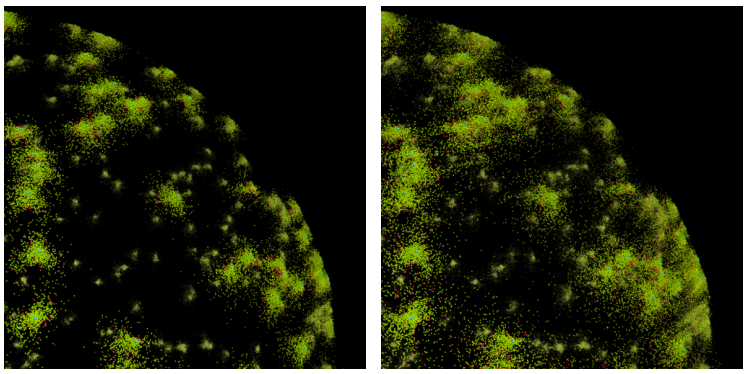
[1] **Parts of this section have been published in**: M. Falk, M. Klann, M. Reuss, and T. Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis 2009)*, pages 169–176, 2009.



(a) (b) (c)

▲ **Figure 6.4** — Drug diffusion in a cancerous tumor visualized with CellVis. (a) entire tumor (1.2 mm in diameter) comprising tumor cells (20 µm, rose) and intercellular fibers (gray). (b) intercellular fibers stabilize the tissue. (c) diffusing drug molecules (20 nm, yellow and red, 15 times enlarged) can attach themselves to cells. Tumor data set courtesy of Michael Klann.

▶ **Figure 6.5** — Particle visualization of diffusing drug molecules. The drug is injected into the tumor at the boundaries. (a) 5 min after injection. (b) 30 min after injection. Tumor data set courtesy of Michael Klann.

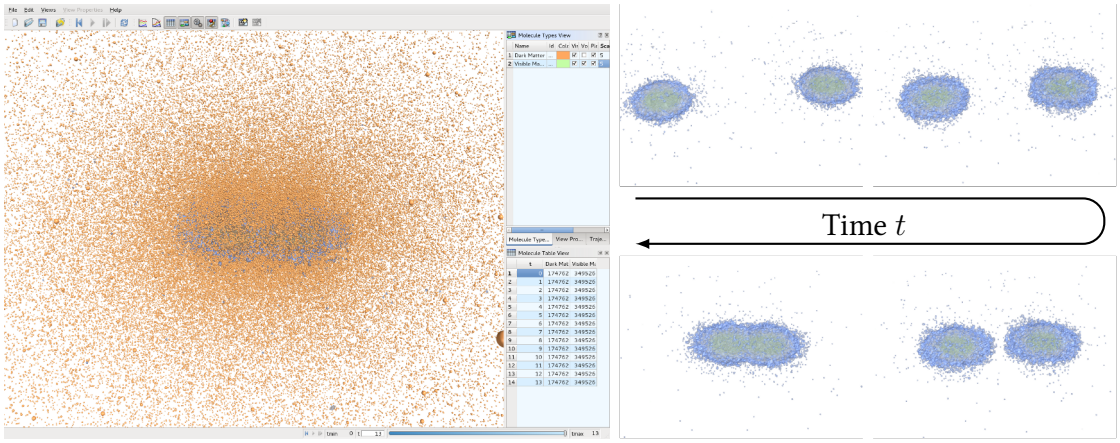


(a) (b)

Figure 6.4 depicts the assembly of the solid tumor and illustrates the different scales. About 110 000 cells and 900 000 fiber elements were used to build the tumor. After thirty minutes roughly 600 000 molecules are diffusing through the tumor or are already bound. However, they reach only cells close to blood vessel endings at the tumor surface. Figure 6.5 shows the states of cells 5 min and 30 min after injection. A realistic tumor, however, can be much larger, thus, increasing the computational complexity and the challenges in optimizing drug delivery.

6.2.3 Colliding Galaxies

Often, particle-based simulations are also used to simulate astronomical phenomena like the formation of galaxies [Springel et al., 2005] or the collision of two or more galaxies [Roettiger et al., 1993]. The galaxies consist of matter which is usually represented by particles. By employing n-body simulations, i.e. each particle interacts with all others,



▲ **Figure 6.6 —** Particle simulation of two colliding galaxies visualized in CellVis. Dark matter is rendered as particles (left, orange) and visible matter is rendered with isosurfaces (left and right). Galaxy data set courtesy of Hendrik Hochstetter.

the forces between the particles can be computed and the particles' movement is then adjusted accordingly.

Figure 6.6 depicts the results of such a simulation. The data set was generated by Hochstetter [2012] employing CUDA. Two galaxies consisting of visible and dark matter collide with each other and fuse together. The left part of Figure 6.6 shows the final time step of the simulation including both types of matter. The temporal development of the galaxies is illustrated on the right of Figure 6.6 with isosurfaces obtained from a density volume of the particles.

CHAPTER 7

RESULTS AND DISCUSSION

This chapter summarizes and discusses the findings of the thesis. The second part highlights the contribution of computer science to systems biology. In particular, the aspects of *in silico* simulations and visualization techniques as tools, which are applied toward the analysis of biological models, are discussed. Finally, a general conclusion integrating the results of the entire thesis is given. Future research directions are outlined in Chapter 8.

7.1 Contribution and Results

In the course of this thesis, methods were developed to support systems biologists in the task of developing, simulating, and visualizing cellular models. The process of model development consists of a cycle of four subsequent steps (cf. Section 1.1). Three of the steps—modeling, simulation, and analysis—are integrated into CellVis, a prototypical framework. CellVis is an integrated, interactive system that supports creating and evaluating cellular models. GPU-based approaches are employed for both, the simulation of the model and the visualization techniques applied during the analysis. The fourth step, i.e. model refinement, is left to the domain experts, but it is also already included in large parts in the modeling and the analysis steps by means of interaction. In the following, the contributions to the individual steps as well as the entire framework are summarized.

7.1.1 Modeling

The basic mesoscopic model of the cell was developed by Michael Klann [Klann, 2011]. It relies on individual particles that represent signaling molecules to model and simulate

cellular signal transduction processes. Due to the discrete and spatial nature of particle-based simulations, this cell model can be more detailed than models based on ODEs and relates, hence, more closely to cells studied in *in vivo* experiments. The cell's interior contains cytoskeletal filaments, the nucleus, signaling molecules, and other proteins. All molecules are approximated by spheres with the respective hydrodynamic radii of the original molecules. Inside the 3D cell model, particles move either by diffusion or along the filaments of the cytoskeleton by motorized transport. Chemical reactions between particles take place according to mass action kinetics and Michaelis-Menten kinetics. In addition, particles can interact with cellular structures.

The original model was extended in several ways. Concepts for a dynamic cytoskeleton were proposed, where local concentrations of membrane-bound receptors are computed and used to affect the growth and decay of microtubules. An additional state was introduced for each molecule to model an activation by an extracellular signal source. The signal is implicitly modeled as time dependent function. With the additional state and the extracellular signal, molecules located at the plasma membrane can be activated. Thus, e.g., the signal transduction processes can be triggered. To model the transport of signaling molecules into the nucleus through nuclear pore complexes, the notion of motorized transport was characterized in finer detail. Signaling molecules can attach themselves to nuclear pores and are transported actively to the other side of the nuclear envelope.

A new graphical notation was presented to describe the cellular model and the interactions inside. Graph-based editing is used to set up and alter molecules and their interactions. The model is stored in an XML format, containing information for the simulation as well as for the visualization. The need for a novel description language arose from the fact that spatial constraints and the intracellular architecture, i.e. the cytoskeleton, are not included in the systems biology mark-up language (SBML). Similarly, the existing systems biology graphical notation (SBGN) does not cover interactions between molecules and cellular structures, which are included in the new proposed notation to model, e.g., the motorized transport.

7.1.2 GPU-based Simulation

Parallel approaches were introduced to parallelize the spatio-temporal particle simulation of the mesoscopic cell model. In particular, the algorithms were designed to fit the many-core architectures of today's GPUs and CPUs. The CUDA framework was utilized for the GPU-based implementation of the simulation. The particle movements including the interactions with cellular structures feature no interdependency and, thus, were easily parallelizable. However, the parallel computation of chemical reactions between particles had to be distributed over several subpasses to be efficient. For baseline comparisons, a CPU implementation employing OpenMP was created employing the same parallel concepts as used on the GPU.

Reasonable effort was spent to optimize the simulations on both platforms, CPU and GPU. There is, however, great potential left for optimization on both, like, e.g., optimizing

memory layouts, considering memory access patterns, and cache coherence. On the CPU, OpenMP itself might be responsible for a comparatively high overhead for managing the threads during the various stages of a single simulation step. The process of parallelization with OpenMP is, however, straight forward and allows one to partly reuse the parallelized and optimized versions of the GPU implementation. The programming of GPUs is simplified by the CUDA environment though the graphics hardware may pose many limitations. For example, the kernel performing the second-order reactions requires a large number of registers thereby limiting the number of available concurrent threads per block.

Despite the efforts of developing common parallel concepts applicable to both GPU and CPU, efforts are still necessary to harness the computational power of the respective hardware. The open compute language (OpenCL) might be one way to overcome the issues between different platforms and architectures, but it will not necessarily be the best choice when maximum possible performance is needed.

Regarding the results in Section 4.5.2, it remains questionable whether so much effort has to be spent on the implementation for a specialized hardware, when simply running the simulation twice on the CPU greatly reduces the benefits of a GPU simulation. Especially if multiple simulation runs are necessary to either account for stochastic effects or to perform parameter studies, it might be simpler to start two or more simulations in parallel on the CPU instead of spending time on further GPU optimizations.

7.1.3 Visualization

For the visualization of the simulation results, GPU-based rendering techniques like glyphs and volume rendering are employed. Novel visualization approaches were developed and applied to the cellular data sets.

The schematic visualization renders the data in the same way as it is used in the simulation. Diffusing particles are represented by spherical glyphs and the filaments of the cytoskeleton are shown as elongated cylinders. Since the data representation matches the actual simulation state of the respective time step, it can provide insight into the mechanisms of the simulation. Although the data of the mesoscopic cell model contains only spherical particles it can be visualized including details on the atomic scale. In the atomistic visualization, each molecule is replaced with its atomic structure. The cytoskeletal filaments are replaced with monomers of the constituting proteins, i.e. tubulin for microtubules and actin for microfilaments. Ray casting techniques are employed to interactively display 25 billion atoms of a single, mesoscopic cell model. Images generated with the atomistic visualization look very similar to the illustrations of David Goodsell [Goodsell, 2009] when applying silhouettes and toon shading.

Ambient occlusion techniques, different shading models, and depth cues like desaturation and DoF are used to reduce the depth complexity and the visual clutter of the intracellular crowding caused by particles and cellular structures. Particle clutter in 3D also complicates the visual tracking of the signaling front, i.e. following activated signaling

molecules. Computing a volumetric representation of the particles and rendering it with DVR techniques makes the progress of the signal much easier to follow.

Two approaches were presented for generating and visualizing volumetric representations of particle data. The first approach relies on a uniform grid enclosing all particles in object space. The density contribution of each particle, computed with a radial-symmetric kernel function, is accumulated in the grid and volume ray casting is used for the subsequent visualization. Since this approach is resolution limited because of the high memory footprint of the volume, the second approach uses an on-the-fly strategy to provide more details while consuming less memory. Volume ray casting is directly coupled with the evaluation of the density field given by the particles and the density field is only evaluated at the sampling positions of the ray casting. Thus, sampling artifacts are reduced and the virtual volume resolution matches the viewport resolution. Hardware occlusion queries and early ray termination are used to prevent unnecessary ray computations. Besides the application in cellular signaling, the approaches have been applied to track protein cavities and to the visualization of particle data sets resulting from MD simulations.

The visualization approach presented last, the microscopic visualization, is targeted for comparing *in silico* models with experimental results. In wet lab experiments, fluorescent markers are often used to tag proteins and are imaged, e.g., with CLSM. Applying a radial intensity distribution to every particle of the cellular model yields images coarsely corresponding to the ones obtained by CLSM and, thus, eases the comparison between both.

7.1.4 CellVis Framework

CellVis provides a simulation interface and an integrated data analysis with focus on visualization. Particle simulations can be accessed by means of computational steering for online data analysis and parameter adjustment during the simulation. Currently, CellVis is used by several researchers for simulation and offline analysis. The primary use is to set up new models and analyze the simulation results. The simulations are usually performed prior to the analysis and computational steering is not employed so far. This might be because computational steering approaches were not in common in biology in the past and researchers are still used to perform offline simulations. Even fast simulations of very small systems, which terminate after a couple of minutes, are analyzed afterward instead of an online simulation where parameter tweaking would be possible. However, for very large models requiring several hours of computing, computational steering seems inappropriate despite being feasible.

Different visualization techniques in CellVis, besides being used for in-depth analysis and interactive exploration, are employed for creating illustrations and also for visual debugging of simulation results. In particular, the analysis of ligand-receptor clustering is benefiting from the glyph-based visualization of bounds since the traditional visualization approach included only the molecule positions shown in a scatter plot.

Multiple coordinated views allow to show cellular data sets from various perspectives with different visualizations. For example, the schematic visualization can be used in the main view whereas the atomistic visualization can provide additional details in a second view. Although the investigation of cellular signal transduction processes is the main application of the prototypical framework, it can also be used for visualizing general particle data sets.

7.2 Visualization and Simulation as Tools in Systems Biology

Emerging at the turn of the century, systems biology is a quite young field of research. Systems biology is a very broad, interdisciplinary field. Research efforts in biology are combined with those in physics, chemistry, engineering, and computer science. Models are created and often evaluated with simulations. The results are then analyzed before the underlying model is refined based on observations in experiments. Thus, it is of high importance to be able to match *in silico* outcomes of simulations with wet lab experiments. Methods known from computer science can contribute to this task.

In silico simulations are a powerful tool for model development but abstractions are often necessary since biological systems appearing, e.g., in a cell are too complex or too detailed. This abstraction might lead to mesoscopic simulations, similar to those presented in this thesis, or to consider only a small fraction of the original system under investigation. The resolution of simulations can be much higher than any experimental resolution, both in the time and the space domain, so simulations cannot only be compared to experiments but, given the right parameters, leading to deeper insights. However, different limitations can occur when employing simulations. Usually not every detail of a biological system is known. The missing information might include unknown chemical reaction rate constants, diffusion coefficients inside compartments, or just the number of signaling molecules involved. This applies in particular to hypothetical, theoretical models when an effect is observed in experiments but its interrelations in the whole system are not well understood. In such cases, parameter estimation and model identification have to be performed first.

In many cases, *in silico* simulations are cheaper and easier to perform than *in vivo* and *in vitro* experiments in the wet lab. Additionally, computer simulations are repeatable with the exact same boundary conditions and, thus, the results can be reproduced more easily. However, standards for the publication of simulation results have yet to be developed. Similar to the approach of providing the experimental setup and the course of action for wet lab experiments, the simulation setup and respective parameters should be given along with the *in silico* results for better reproducibility.

The model setup and the analysis of the simulation results require the expertise of domain scientists, i.e. biologists. Computer science can provide the means and techniques for the simulation but not the necessary biological background. Methods for accelerating or

improving existing code can be applied to increase the overall efficiency of the simulations. The application of GPGPU techniques to parallelize the simulation as demonstrated in this thesis is just one example. In addition, approaches from software engineering can help to improve the development process of simulation codes. This could ensure and increase the maintainability and expandability of the simulations.

With more powerful simulations, the underlying models will presumably get larger and more detailed soon. To understand the underlying mechanisms, a model containing more details is, however, not always beneficial. Consider simulating the entire human body to perform, for instance drug sensitivity analysis. If every single cell of the body would be modeled at the highest available level of detail, a simulation would simply not be feasible because it would take too much time to compute. Additionally, the fate of most individual cells would be irrelevant since they are not affected by the drugs. The overall outcome of the simulation would most likely be precise for the given parameter set. However, the actual effect the drug has on a signaling pathway inside a cell might be lost in the statistical noise of all cells. The level of detail of the simulation raises another question—the trade-off between details and time required. Systems of ODEs or PDEs are efficient, but spatial resolution is limited; whereas particle-based simulations require more computational power while providing spatial details. By coupling these two approaches, it should be possible to model and simulate the drug sensitivity analysis described above in a reasonable amount of time. Despite simulating the model on the whole scale, the drug effects on the single cell level are still included.

After the biological model has been simulated, the results have to be analyzed. The application of visualization approaches is not yet widely spread in systems biology, or biology at large. At most, graphs and simple volumetric renderings are the only visualizations in addition to the statistical data currently preferred by domain experts for analysis purposes. Two-dimensional and three-dimensional graphical representations are usually employed for illustrative reasons only, but not for interactive, visual analysis. Simulation results are often analyzed by generating several plots and studying them. Thus, spatial effects, which are inherent to the solutions of PDEs or particle-based simulations, are typically not considered.

In the last years, the long neglected part of visualization in biology attracted the visualization community more and more. The distinct research areas of biologists and visualization researchers began to merge. Recent events like *BioVis*[1] and *Vizbi*[2] are just two examples of the efforts that are undertaken to bring these two worlds together, thereby stimulating collaborative research. Methods of the scientific visualization community, like the ones presented in this thesis, are gradually accepted by biologists. Techniques known from information visualization, like scatter plots and parallel coordinates, also find their way into systems biology. Besides the interactive visualization, the approach of visual analytics is also applicable [e.g. Heinrich et al., 2012].

[1] IEEE Symposium on Biological Data Visualization. http://www.biovis.org
[2] Visualization in Biology. http://www.vizbi.org

When developing new visualization techniques for biological applications, it is important to consider input of biologists working on specific problems. The visualization researcher will often find biologists that are open to new ideas. But sometimes biologists have very definite point of views like, e.g., that the background color of the visualization has to be dark. Such opinions have to be considered and discussed. However, new methodologies should not be developed exclusively for the visualization of *in silico* simulation results. They should also support the incorporation of actual experimental data. Thus, visualization approaches will become valuable tools, which will make the otherwise distinct boundary between simulation and experiment permeable.

7.3 Conclusion

In this thesis, GPU-based approaches have been developed for and applied to the context of systems biology. Intracellular signal transduction processes, i.e. the transport of signaling molecules through the intracellular environment, were chosen as the underlying model mechanisms of interest. A simplified, mesoscopic model of the cell serves as simulation environment for the signal transduction. The cellular model consists of the plasma membrane, the nucleus, filaments of the cytoskeleton, and a variety of signaling molecules.

In order to accelerate the cellular particle simulation, concepts for the parallelization of the simulation were presented, especially for particle interactions and particle movement by diffusion. A simulation prototype was implemented in CUDA to employ the parallel many-core architecture of recent GPUs. The performance of the parallel, GPU-based simulation was evaluated in several scenarios and compared to a reference implementation running on the CPU. Employing the GPU, the overall speedup factor ranged from 1 to 10 depending on the scenario. For a realistically sized MAP3K pathway, the speedup was about 3.5. However, the question remains whether it is beneficial to painstakingly optimize code to execute on a dedicated hardware like the GPU or expend the same amount of time optimizing a more generic and maintainable CPU code.

For the analysis of the particle-based data resulting from the simulation, novel GPU-based visualization techniques are proposed. By combining approaches from glyph-based rendering and DVR, the molecules in the cellular environment can be visualized as discrete glyphs or as a continuum, which represents the local molecule concentrations. In addition to the glyph-based visualization, the components of the mesoscopic, cellular model can be enriched by incorporating and visualizing their respective atomic structures. The atomistic visualization employs a hierarchical ray casting to display several billion of atoms at interactive rates and can be interpreted as a first step toward multiscale visualization. The microscopic visualization generates images that are highly similar to images obtained by CLSM. Using this visualization, biologists can compare the outcome of the *in silico* simulations with wet lab experiments.

Simulation and visualization have been combined into a new prototypical framework. Computational steering of the simulation is possible by a tight coupling of both components. The major application area of the framework is the simulation and visualization of signal transduction pathways. However, the framework is also applicable to the visualization and investigation of generic particle-based data as demonstrated in this thesis.

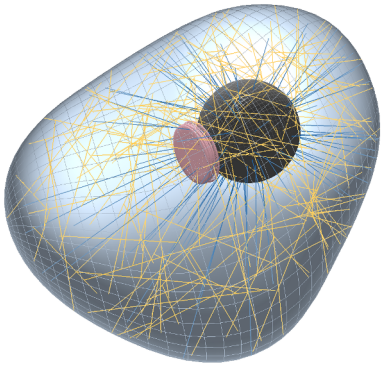
In the context of systems biology, especially in that of the process of model development and validation, both *in silico* simulations and visualizations are useful tools. In addition, methods from computer science and visualization can help to improve existing approaches or develop new methods driven by the requirements of systems biologists. However, a close cooperation between computer scientists and biologists is nonetheless required because computer scientists are no field experts in biology and vice versa. The work presented in this thesis is a result of one such successful, interdisciplinary collaboration.

CHAPTER 8

OUTLOOK

In this thesis, the basic principles for creating, simulating, and visualizing a cellular model were presented. These principles could serve as a starting point for future research. In its current state, the prototypical implementation of CellVis includes a particle-based simulation of a simplified cellular model and provides means for the subsequent analysis by visualizing results of simulations. The prototype is already used by several biologists and received positive feedback. By adding new features to the individual parts of the framework, CellVis can be adjusted to the needs of domain experts. Future directions of research ranging from the refinement of the analysis stage to the simulation of drug medications at the scale of the human body are outlined in the following.

The techniques proposed in this thesis are well-suited for the analysis of individual data sets. Biologists, however, usually require multiple simulation runs of the same model due to the stochastic nature of the simulation to account for probabilistic effects. The simulations are performed either with a slightly varying parameter set or an identical parameter set but with different random seeds. Variations of the parameters like the number of molecules leads to extrinsic noise whereas the stochasticity of the modeled signal pathway yields intrinsic noise [Swain et al., 2002]. A statistical evaluation of all the results should ease the comparisons between all simulation runs. By comparing the first few statistical moments, various simulation runs can be quickly compared. However, this can pose problems in some cases because the statistical values only describe some inherent characteristics of the data. Important features like, e.g., the placement of the cytoskeleton might render irrelevant in the statistical analysis despite having impact on particle transport. Thus, it is necessary to also consider the actual, geometric intracellular architecture of the cell when performing comparisons based on the statistical nature of the data set. To incorporate the results of the statistical analysis into the visualization, they could be combined with the existing visualizations approaches. One data set could be selected as ground truth data and the deviation between ground truth



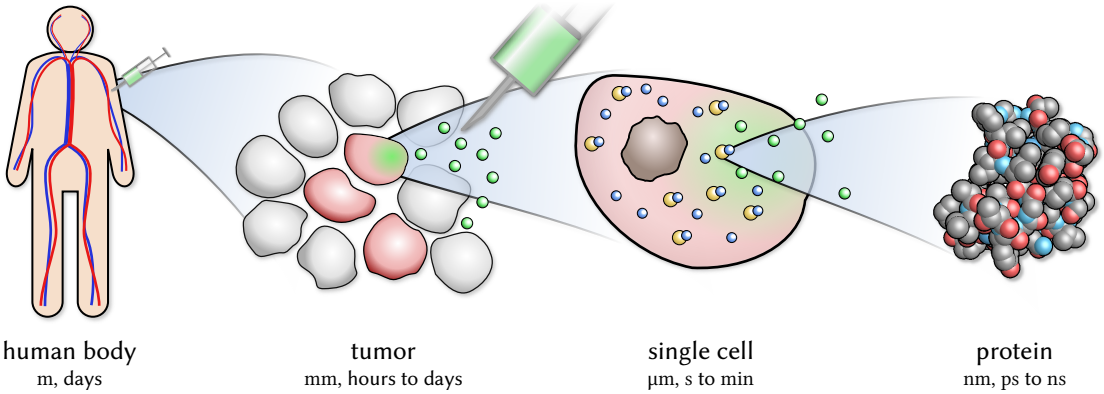
◀ **Figure 8.1 —** Cell featuring a deformed plasma membrane. The cellular interior comprises the nucleus (gray), microtubules (blue), microfilaments (yellow), and a simplified model of the Golgi apparatus (red).

and the results of additional simulation runs could be visualized. This implies, however, that the dimensions of the simulations have to be identical with that of the ground truth data. Hence, additional visualization approaches will be necessary to compare data of different time and length scales. Approaches like World Lines [Waser et al., 2010] or VisTrails [Silva et al., 2007] might be able to support the model development process when comparing different parameter settings.

As mentioned previously, the utilization of the GPU for the simulation is often beneficial except for some special cases. Maintaining two simulation codes, one for the GPU and another one for the CPU, is too expensive. Thus, it makes sense to focus on only one target platform. Following the trends of hardware development of GPUs in recent years, the computational performance is increasing at a much higher rate than that of CPUs. Although the many-core architecture of the GPU is not perfectly suited for the cellular particle simulation, this path should, hence, be taken. Another possibility would be the usage of a compute cluster where the simulation is either distributed over several nodes or each node runs an individual simulation. Since not every biologist has access to a compute cluster, it would make sense to run the simulations on the same desktop computer where the visualizations are applied and the user performs the analysis. Another potential solution to that problem might be cloud computing approaches where data centers provide the hardware and handle the computations. One has to bear in mind that the simulation creates huge amounts of data, which have to be transferred back to the user before the analysis step can take place.

To reduce the effort needed to set up cellular models in the long term, the modeling component might be extended with rule-based modeling, e.g., with the Kappa Language [Danos et al., 2008]. Describing complex chemical reactions or the distinction of several phosphorylation sites of a single molecule would be much simpler using rule-based modeling. For example, a molecule with two phosphorylation sites can assume eight different states when explicitly modeled whereas a set of three rules is sufficient when using rule-based modeling. The modeled interaction graph could then be generated from a given rule set and the simulation would subsequently reuse this interaction graph for evaluation.

In addition to rule-based modeling, it might be beneficial to couple different simulation modalities on the same scale to improve the overall simulation. The particle simulation



▲ **Figure 8.2 —** Multiscale simulation of drug administration. Time and length scales are coupled over four distinct tiers.

could, e.g., be combined with an ODE model so that the models complement each other. In particular, the system of ODEs could be used inside a particular compartment, like the Golgi apparatus, and both models are then linked at the surface of the compartment.

So far, the simplified cell model features a perfectly sphere-shaped plasma membrane but the cells in wet lab experiments rarely display such a shape. The shape of the plasma membrane and the interior cellular structures might be explicitly modeled or obtained from experimental data. In a preliminary work together with Hendrik Hochstetter non-uniform rational B-spline (NURBS) surfaces are used to deform the plasma membrane. Figure 8.1 shows an exemplary cell with its interior. Electron tomography can be used to extract microtubules in wet lab experiments [Weber et al., 2012]. A third approach, proposed by Mosaliganti et al. [2008], reconstructs 3D cellular shapes from 2D images obtained with optical microscopy techniques. CellOrganizer [Murphy, 2012] applies model learning to generate cellular shapes and the cytoskeleton from images. All approaches could be used to affect the cellular shape and subsequently the cytoskeleton inside the membrane. The result is a cellular shape that matches cell images obtained from *in vivo* or *in vitro* experiments. Whether a deformable plasma membrane is a useful extension of the framework remains to be seen. Applying the microscopic visualization to the deformed cell model should yield images similar to the experiments. Thus, the direct comparison between experimental data and the *in silico* results can be simplified. However, comparing the results of different simulations, i.e. different cellular shapes, becomes more complicated due to the irregular shapes.

Eventually it will also be necessary to embed the cell into its *in vivo* or *in vitro* environment such as microfluidic devices or organs in the body. To simulate tissue or even organs consisting of individual cells, multiple tiers are required because of the different scales. A multiscale simulation can incorporate various different scales and is well suited to describe hierarchical systems that occur in biology, like organs or the human body [Bittig and Uhrmacher, 2010]. In Figure 8.2, an exemplary multiscale model of drug administration is illustrated. In a coarse-grained model of the human body, drugs are injected into the

blood circle and are transported to a tumor. At the tumor, the drug molecules diffuse through cancerous tissue. The effects of the drugs onto the signaling pathways are studied on single cell level. On the lowest scale, MD simulations could be used to analyze protein behavior like inhibition or cleavage. When switching to a more detailed model, both the length scale and the temporal scale are adjusted accordingly—with special care at the interface between the different scales. The multiscale simulation can be set up as top-down approach, beginning at the coarse-grained scale and refining down to the proteins, as bottom-up approach, or a combination of both. In particular, this multiscale system can be evaluated and analyzed using a computational steering approach. Only the scale at which the user is currently interested in is simulated with full detail and the remaining tiers could be approximated. As soon as the user changes the current scale, the simulation is adjusted accordingly.

Although the coupled multiscale model of drug administration is fiction at the present time, it closely resembles the idea of systems biology—to investigate the characteristics and the complex interactions of all elements in a particular biological system—as stated in Chapter 1. In the future, such models might be used for pharmaceutical drug testing. This could in turn lead to the design of patient-tailored medication directly at the physician requiring only a couple of minutes. The key challenges in such multiscale models, however, are the coupling of the different scales and, from the visualization point of view, the navigation through time and space.

Model Parameters

A.1 ODE Example

The ODE example is a small, exemplary system containing only a single chemical reaction. It contains two molecular species A and B that are uniformly distributed in the simulation domain. They react with each other in the reaction $A + B \longrightarrow A$. Thereby, the concentration of A should remain constant during the simulation whereas the concentration of B should decrease. Parameters for this model configuration are given in Table A.1.

▼ **Table A.1 —** Parameters of the ODE example.

Simulation Domain
$r_{\text{membrane}} = 2\,\mu\text{m}$
$r_{\text{nucleus}} = 0.6\,\mu\text{m}$
$V = \frac{4}{3}\pi\left(r^3_{\text{membrane}} - r^3_{\text{nucleus}}\right) = 32.606\,\mu\text{m}^3$
Reactions
$A + B \xrightarrow{k_{AB}} A$
$k_{AB} = 1 \times 10^7\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$
Molecule Counts
$A = 1947$
$B = 1947$
Molar Concentrations
$c_A = \frac{1947}{V \cdot 10^{-15} N_A} = 1 \times 10^{-7}\,\text{mol}\,\text{L}^{-1}$
$c_B = 1 \times 10^{-7}\,\text{mol}\,\text{L}^{-1}$

A.2 Enzymatic Reaction

In an enzymatic reaction, an enzyme E converts the substrate S into a new product P by creating an intermediary complex ES. Such reactions can be described by Michaelis-Menten kinetics [Michaelis and Menten, 1913]. Applying Michaelis-Menten kinetics, the enzymatic reaction of $E + S \rightleftharpoons ES \longrightarrow E + P$ can be simplified to $E + S \longrightarrow E + P$ (cf. Section 3.1.6). With this model, two possible configurations are tested. First, a improperly defined configuration $c_E > K_m$, where the steady-state assumption does not hold. The second contains a parameter set fulfilling the assumption for the steady-state, i.e. $c_E \ll K_m$. In Table A.2, the respective parameters are given.

▼ **Table A.2 —** Parameters of the enzymatic reaction example.

	$c_E > K_m$	$c_E \ll K_m$
c_E	$20\,\mathrm{nmol\,L^{-1}}$	$10.7\,\mathrm{nmol\,L^{-1}}$
K_m	$9.9\,\mathrm{nmol\,L^{-1}}$	$30\,\mathrm{nmol\,L^{-1}}$
k_{cat}	$1.89\,\mathrm{s^{-1}}$	$0.25\,\mathrm{s^{-1}}$
τ_E	0.53 s	4 s

A.3 Benchmark Parameters

The benchmark data set was modeled to measure different effects on the overall performance. The following effects were studied with this model: weak and strong scalability (Sections 4.4.1, 4.4.1), a varying number of particles (Section 4.4.2), and the influence of the size of the acceleration grids (Section 4.4.3). The baseline of the data set was determined to be at 50 000 particles after some preliminary tests. At this point, the CPU simulation is slightly outperformed by the GPU version. The model of the baseline configuration is given in Table A.3. The modeled system contains one first-order reaction ($D \longrightarrow E$) and one of second order ($A + B \longrightarrow A + C$). The reactions are modeled in a way that the overall number of particles is constant during the whole simulation.

Two scenarios, both maintaining a constant rate of reacting particles per simulation step, are used. In the first scenario, the scene size, i.e. the radii of plasma membrane and nucleus, remains constant while the molecule concentration scales with the number of particles. Hence, the reaction rates have to be scaled accordingly to maintain a constant rate of performed reactions per simulation step. First-order reaction rate constants depend linearly on the number of particles whereas second-order reactions have to be scaled quadratically. The second scenario, the growing scene, leaves the reaction rate constants unchanged, but modifies the radii of the plasma membrane and the nucleus in turn. The radii are adjusted so that the global concentration remains equal to the baseline configuration. In addition, the number of cytoskeletal filaments is modified to maintain a constant occupied volume fraction of 1.5 %. The respective parameters are given in Table A.3.

▼ **Table A.3 —** Parameters of the benchmark setup. Molecule numbers are given for the baseline configuration of 50 000 particles.

	Const. Scene Size	Growing Scene
Simulation Domain		
r_{membrane}	2 µm	4.08 µm
r_{nucleus}	0.6 µm	1 µm
$\Delta t = 2 \times 10^{-6}$ s		
Steps $n = 50\,000 = 50\,000 \times 1$ (interval × frames written)		
Grid resolution $30 \times 30 \times 30$		
Molecules and Structures		
A ($n = 16\,666$), B ($n = 16\,666$), D ($n = 16\,668$), C ($n = 0$), E ($n = 0$)		
$r = 0.003$ µm		
$D = 3 \times 10^{-12}$ m^2/s		
Microtubules, radial placement		
$r = 0.0125$ µm		
n	500	2000
l	1 µm to 3 µm	3 µm to 5 µm
Microfilaments, random placement		
$r = 0.0004$ µm		
n	7000	10 800
l	0.05 µm to 1 µm	1 µm to 4 µm
Reactions		
$A + B \xrightarrow{k_{AB}} A + C$		
$D \xrightarrow{k_D} E$		
k_{AB}	1.36×10^5 mol L^{-1} s^{-1}	1×10^7 mol L^{-1} s^{-1}
k_D	0.117 s^{-1}	1 s^{-1}
Initial Molar Concentrations		
c_A, c_B, c_D	8.49×10^{-7} mol L^{-1}	1×10^{-7} mol L^{-1}

A.4 Simplified MAPK Example

In this model, the three tiers of the MAPK pathway (cf. Section 2.1.2) are reduced to a single tier. MAP2K and MAP3K, both belonging to the upstream part of the pathway, are replaced with a single receptor complex. This complex is positioned at the plasma membrane of the cell. Hence, the MAPK is directly phosphorylated, i.e. activated, by the membrane-bound receptor complex. The deactivation of the signal by dephosphorylating the activated MAPK molecules is also included. The necessary simulation parameters are detailed in Table A.4.

▼ **Table A.4 —** Parameters of the simplified MAPK example.

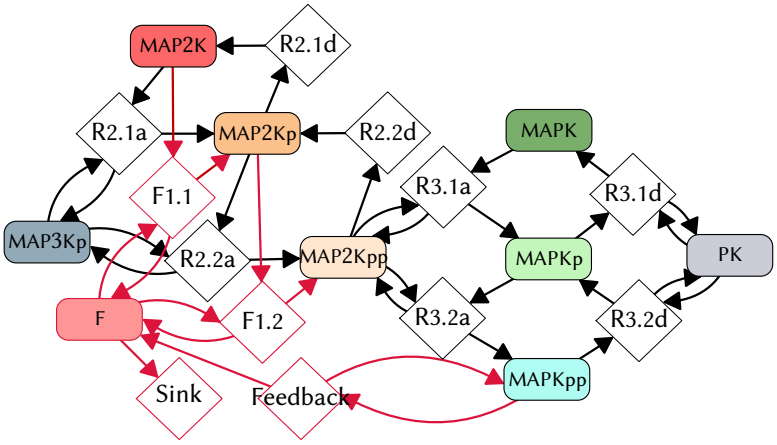
Simulation Domain
$r_{\text{membrane}} = 4.667\,\mu\text{m}$
$r_{\text{nucleus}} = 1\,\mu\text{m}$
$\Delta t = 9 \times 10^{-7}\,\text{s}$
Steps $n = 2\,000\,000 = 10\,000 \times 200$ (interval × frames written)
Grid resolution $32 \times 32 \times 32$
Molecules and Structures
Membrane-bound receptors (R, $n = 5000$)
$r = 0.005\,\mu\text{m}$
$D = 1 \times 10^{-14}\,\text{m}^2/\text{s}$
MAPK ($n = 35\,000$)
$r = 0.01\,\mu\text{m}$
$D = 1 \times 10^{-12}\,\text{m}^2/\text{s}$
MAPKp ($n = 0$)
$r = 0.011\,\mu\text{m}$
$D = 1 \times 10^{-12}\,\text{m}^2/\text{s}$
P_+, activate phosphatase ($n = 10\,000$)
$r = 0.005\,\mu\text{m}$
$D = 1.5 \times 10^{-11}\,\text{m}^2/\text{s}$
P_-, inactive phosphatase ($n = 0$)
$r = 0.005\,\mu\text{m}$
$D = 1.5 \times 10^{-11}\,\text{m}^2/\text{s}$
Crowding molecules ($n = 20\,000$)
$r = 0.015\,\mu\text{m}$
$D = 0\,\text{m}^2/\text{s}$
Microfilaments ($n = 5000$), random placement
$r = 0.004\,\mu\text{m}$, $l = 0.5\,\mu\text{m}$ to $2\,\mu\text{m}$
Microtubules ($n = 600$), radial placement
$r = 0.0125\,\mu\text{m}$, $l = 2\,\mu\text{m}$ to $5\,\mu\text{m}$
Reactions
$\text{MAPK} + \text{R} \xrightarrow{k_{\text{act}}} \text{MAPKp} + \text{R}$
$\text{MAPKp} + \text{P}_+ \xrightarrow{k_{\text{deact}}} \text{MAPK} + \text{P}_-$
$\text{P}_- \xrightarrow{k_{\text{p}}} \text{P}_+$
$k_{\text{act}} = 1 \times 10^8\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$
$k_{\text{deact}} = 3 \times 10^8\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$
$k_{\text{p}} = 20\,\text{s}^{-1}$
Interactions
Motorized transport along microtubules
$\text{MAPKp} \overset{k_f}{\longleftrightarrow} \text{MAPKp}^*$
$k_{f,bind} = 20\,000\,\text{s}^{-1}$

▼ **Table A.4 —** Parameters of the simplified MAPK example. (continued)

$k_{f,unbind} = 500\,\mathrm{s}^{-1}$
$v_f = 10^{-5}\,\mathrm{m\,s}^{-1}$
Attachment to nucleus
MAPKp $\overset{k_n}{\longleftrightarrow}$ Target
$k_{n,bind} = 20\,000\,\mathrm{s}^{-1}$
$k_{n,unbind} = 0\,\mathrm{s}^{-1}$

A.5 MAP3K Example

In contrast to the model described in the previous section, the MAP3K example includes all three stages including MAP3K, MAP2K, and the final tier MAPK. Figure A.1 shows the setup of the model, which is based on the model of Markevich et al. [2006]. Table A.5 shows the initial number of molecules and the cellular setup. In total, about 246 000 molecules have to be moved in each step of the simulation. Up to 75 000 molecules are additionally created in the feedback loop. The reactions and reaction rate constants are given in Table A.6 for first-order and second-order reactions.



▲ **Figure A.1 —** Reaction network of the generalized MAPK pathway model. The feedback loop for the traveling wave is highlighted in red. Reactions (white diamonds) are denoted with *a* for an activation and *d* for a deactivation.

▼ **Table A.5 —** Parameters of the generalized MAP3K pathway.

Simulation Domain
$r_{\text{membrane}} = 5\,\mu\text{m}$
$r_{\text{nucleus}} = 2\,\mu\text{m}$
$\Delta t = 7.5 \times 10^{-5}\,\text{s}$
Steps $n = 160\,000 = 10\,000 \times 200$ (interval × frames written)

▼ **Table A.5 —** Parameters of the generalized MAP3K pathway. (continued)

Grid resolution 50 × 50 × 50
Molecules and Structures
MAP3Kp (20 000), MAP2K (60 000), MAP2Kp (0), MAP2Kpp (0), MAPK (106 000), MAPKp (0), MAPKpp (0), Phosphatase (PK, 60 000), Feedback (F, 0)
$r = 0.02\,\mu m$
$D = 2 \times 10^{-12}\,m^2/s$
Microfilaments (800), random placement
$r = 0.01\,\mu m$, $l = 0.2\,\mu m$ to $1.5\,\mu m$
Microtubules (400), radial placement
$r = 0.025\,\mu m$, $l = 0.5\,\mu m$ to $3\,\mu m$

▼ **Table A.6 —** Reactions and rate constants of the generalized MAP3K pathway.

No.	Second-order Reaction	k_{ij} [L/(mol s)]	τ [s]
R2.1 act	MAP3Kp + MAP2K ⟶ MAP3Kp + MAP2Kp	7.69×10^6	1.0
R2.2 act	MAP3Kp + MAP2Kp ⟶ MAP3Kp + MAP2Kpp	1.00×10^8	0.2
R3.1 act	MAP2Kpp + MAPK ⟶ MAP2Kpp + MAPKp	2.00×10^7	1.0
R3.1 deact	PK + MAPKp ⟶ PK + MAPK	1.90×10^8	0.53
R3.2 act	MAP2Kpp + MAPKp ⟶ MAP2Kpp + MAPKpp	4.00×10^8	0.05
R3.2 deact	MAPKpp + PK ⟶ MAPKp + PK	1.39×10^7	4.00
Feedback	2 MAPKpp ⟶ 2 MAPKpp + F	1.00×10^8	–
F1.1	F + MAP2K ⟶ F + MAP2Kp	5.00×10^8	–
F1.2	F + MAP2Kp ⟶ F + MAP2Kpp	5.00×10^8	–

No.	First-order Reaction	k_i [1/s]
R2.1 deact	MAP2Kp ⟶ MAP2K	0.833
R2.2 deact	MAP2Kpp ⟶ MAP2Kp	0.833
Feedb. sink	F ⟶ ∅	64.8

A.6 Nuclear Import and Export

The import and export of molecules through the nuclear envelope is handled by nuclear pores (cf. Section 3.1.7). The model detailed in Table A.7 initially contains two molecular species. The molecules of type C are located in the cytoplasm and are imported into the nucleus in the course of the simulation. Inside the nuclear envelope reside the molecules of type N that are to be exported through the nuclear pores into the cytoplasm.

▼ **Table A.7 —** Parameters of the nuclear import example.

Simulation Domain
$r_{\text{membrane}} = 5\,\mu\text{m}$
$r_{\text{nucleus}} = 1\,\mu\text{m}$
$\Delta t = 7.5 \times 10^{-7}\,\text{s}$
Steps $n = 100\,000 = 1000 \times 100$ (interval × frames written)
Grid resolution $32 \times 32 \times 32$
Molecules and Structures
$\text{C}_{\text{cytoplasm}}$, in cytoplasm ($n = 20\,000$)
$r = 0.03\,\mu\text{m}$
$D = 2 \times 10^{-10}\,\text{m}^2/\text{s}$
$\text{C}_{\text{nucleus}}$ ($n = 0$)
$r = 0.02\,\mu\text{m}$
$D = 5 \times 10^{-12}\,\text{m}^2/\text{s}$
$\text{N}_{\text{cytoplasm}}$ ($n = 0$)
$r = 0.03\,\mu\text{m}$
$D = 2 \times 10^{-11}\,\text{m}^2/\text{s}$
$\text{N}_{\text{nucleus}}$ ($n = 200$)
$r = 0.02\,\mu\text{m}$
$D = 2 \times 10^{-11}\,\text{m}^2/\text{s}$
Microfilaments ($n = 150$), random placement
$r = 0.05\,\mu\text{m}$, $l = 0.8\,\mu\text{m}$ to $1.5\,\mu\text{m}$
Microfilaments ($n = 200$), radial placement
$r = 0.05\,\mu\text{m}$, $l = 2.5\,\mu\text{m}$ to $3\,\mu\text{m}$
Microtubules ($n = 400$), polarized placement
$r = 0.05\,\mu\text{m}$, $l = 2\,\mu\text{m}$
Nuclear Pores ($n = 500$)
$r = 0.012\,\mu\text{m}$
$l = 0.2\,\mu\text{m}$
$\text{C}_{\text{cytoplasm}} \xrightarrow{k_{\text{C}}} \text{C}_{\text{nucleus}}$, $k_{\text{C}} = 1\,000\,000\,\text{s}^{-1}$
$v_{\text{C}} = 0.0006\,\text{m}\,\text{s}^{-1}$
$\text{N}_{\text{nucleus}} \xrightarrow{k_{\text{N}}} \text{N}_{\text{cytoplasm}}$, $k_{\text{N}} = 800\,000\,\text{s}^{-1}$
$v_{\text{N}} = -0.0006\,\text{m}\,\text{s}^{-1}$

A.7 Dynamic Cytoskeleton

In this model, the cell responds to an extracellular signal by elongating the cytoskeletal filaments. The growth and shrink rates of the cytoskeleton are exaggerated so that the growth effect becomes visible in half a second of simulated time. A single pulse is sweeping over the cellular membrane thereby activating membrane-bound receptors (cf. Section 3.1.8). The activated receptors contribute to the local concentrations, which in

turn affect the growth of the cytoskeleton as described in Section 3.1.9. The respective parameter set is given in Table A.8.

▼ **Table A.8 —** Parameters of the dynamic cytoskeleton example.

Simulation Domain
$r_{\text{membrane}} = 4\,\mu\text{m}$
$r_{\text{nucleus}} = 1\,\mu\text{m}$
$\Delta t = 2 \times 10^{-5}\,\text{s}$
Steps $n = 24\,000 = 200 \times 120$ (interval × frames written)
Grid resolution $30 \times 30 \times 30$
Molecules and Structures
Membrane-bound receptors (R, $n = 5000$)
$r = 0.04\,\mu\text{m}$
$D = 1 \times 10^{-11}\,\text{m}^2/\text{s}$
A ($n = 20\,000$), $\text{A}_{\text{activated}}$ ($n = 0$)
$r = 0.04\,\mu\text{m}$
$D = 2 \times 10^{-11}\,\text{m}^2/\text{s}$
Microtubules ($n = 500$), radial placement
$r = 0.0125\,\mu\text{m}$, $l = 1\,\mu\text{m}$ to $2.9\,\mu\text{m}$
Reactions
$\text{R} + \text{A} \xrightarrow{k_{\text{RA}}} \text{R} + \text{A}_{\text{activated}}$, $k_{\text{RA}} = 1 \times 10^{23}\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$
Cytoskeleton Dynamics
$v_{\text{grow}} = 18.28\,\mu\text{m}\,\text{s}^{-1}$
$v_{\text{shrink}} = -5.8\,\mu\text{m}\,\text{s}^{-1}$
$v_{\text{catastr}} = -20.5\,\mu\text{m}\,\text{s}^{-1}$
$c_{\text{grow}} = 12.5\,/\mu\text{m}^2$
$c_{\text{shrink}} = 0.2\,/\mu\text{m}^2$
Stimulus Function $f(t, \boldsymbol{p})$
$t_0 = 0\,\text{s}$
$k_+ = 1000\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$, $k_- = 700\,\text{mol}\,\text{L}^{-1}\,\text{s}^{-1}$
$w = 1\,\mu\text{m}$
$d_p = \infty$
$v = 3 \times 10^{-5}\,\text{m}\,\text{s}^{-1}$
$c_{\text{high}} = 30\,/\mu\text{m}^2$
$c_{\text{low}} = 0\,/\mu\text{m}^2$

Hardware Configuration

Name	Details
CPU	Intel Core i7 Dual Core 3 GHz 8 GiB RAM
GPU	NVIDIA GeForce 670 GTX 2 GiB RAM
Operating system	Windows and Linux
Viewport resolution	1600 × 1200

Bibliography

I. Alduán and M. A. Otaduy. SPH granular flow with friction and cohesion. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 25–32, 2011. [page 22]

M. D. Allen, L. M. DiPilato, B. Ananthanarayanan, R. H. Newman, Q. Ni, and J. Zhang. Dynamic visualization of signaling activities in living cells. *Science Signaling*, 1(37): pt6, 2008. [page 17]

J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EuroGraphics 1987*, pages 3–10, 1987. [page 106]

J. A. Anderson, C. D. Lorenz, and A. Travesset. General purpose molecular dynamics simulations fully implemented on graphics processing units. *Journal of Computational Physics*, 227(10):5342–5359, 2008. [pages 18 and 23]

S. S. Andrews and D. Bray. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Physical Biology*, 1(3):137–151, 2004. [page 53]

S. S. Andrews, N. J. Addy, R. Brent, and A. P. Arkin. Detailed simulations of cell biology with Smoldyn 2.1. *PLoS Computational Biology*, 6(3):e1000705, 2010. [pages 24 and 53]

B. R. Angermann, F. Klauschen, A. D. Garcia, T. Prustel, F. Zhang, R. N. Germain, and M. Meier-Schellersheim. Computational modeling of cellular signaling processes embedded into dynamic spatial contexts. *Nature Methods*, 9(3):283–289, 2012. [page 19]

C. L. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. TexMol: Interactive visual exploration of large flexible multi-component molecular complexes. In *IEEE Visualization 2004*, pages 243–250, 2004. [page 35]

L. Bavoil and M. Sainz. Image-space horizon-based ambient occlusion. In W. Engel, editor, *ShaderX*^{7}, chapter 6.2, pages 425–444. Charles River Media, 2009. [page 93]

E. Bertini and D. Lalanne. Investigating and reflecting on the integration of automatic data analysis and visualization in knowledge discovery. *ACM SIGKDD Explorations Newsletter*, 11(2):9–18, 2010. [page 3]

E. Betzig, G. H. Patterson, R. Sougrat, O. W. Lindwasser, S. Olenych, J. S. Bonifacino, M. W. Davidson, J. Lippincott-Schwartz, and H. F. Hess. Imaging intracellular fluorescent proteins at nanometer resolution. *Science*, 313(5793):1642–1645, 2006. [page 18]

A. T. Bittig and A. M. Uhrmacher. Spatial modeling in cell biology at multiple levels. In *Winter Simulation Conference*, pages 608–619, 2010. [page 151]

J. F. Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics (Proceedings of SIGGRAPH 1977)*, 11(2):192–198, 1977. [page 90]

J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982. [pages 35 and 116]

M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004. [page 60]

M. Branschädel, A. Aird, A. Zappe, C. Tietz, A. Krippner-Heidenreich, and P. Scheurich. Dual function of cysteine rich domain (CRD) 1 of TNF receptor type 1: Conformational stabilization of CRD2 and control of receptor responsiveness. *Cellular Signalling*, 22(3): 404–414, 2010. [page 14]

B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization*, pages 91–98, 1994. [page 41]

N. A. Campbell. *Biology*. Benjamin-Cummings Publishing Company, Inc., fourth edition, 1996. [pages 7 and 174]

H. Casanova, F. Berman, T. Bartol, E. Gokcay, T. Sejnowski, A. Birnbaum, J. Dongarra, M. Miller, M. Ellisman, M. Faerman, G. Obertelli, R. Wolski, S. Pomerantz, and J. Stiles. The virtual instrument: Support for grid-enabled MCell simulations. *International Journal of High Performance Computing Applications*, 18(1):3–17, 2004. [pages 24 and 133]

J. M. Cecilia, J. M. García, G. D. Guerrero, M. A. M. del Amor, I. Pérez-Hurtado, and M. J. Pérez-Jiménez. Simulation of P systems with active membranes on CUDA. *Briefings in Bioinformatics*, 11(3):313–322, 2010. [page 64]

D. Cha, S. Son, and I. Ihm. GPU-assisted high quality particle rendering. *Computer Graphics Forum*, 28(4):1247–1255, 2009. [page 117]

D. Chandran, F. T. Bergmann, and H. M. Sauro. Tinkercell: modular CAD tool for synthetic biology. *Journal of Biological Engineering*, 3(1):19–36, 2009. [pages 22 and 133]

V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling, symmetries, refinements. In J. Fisher, editor, *Formal Methods in Systems Biology*, volume 5054 of *Lecture Notes in Computer Science*, pages 103–122. Springer Berlin / Heidelberg, 2008. [pages 60 and 150]

J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *ACM Workshop on Volume Visualization*, pages 91–98, 1992. [page 44]

L. Dematté. Parallel particle-based reaction diffusion: a GPU implementation. In *Workshop on High Performance Computational Systems Biology*, pages 67–77, 2010. [pages 24 and 71]

L. Dematté and T. Mazza. On parallel stochastic simulation of diffusive systems. In *International Conference on Computational Methods in Systems Biology (CMSB 2008)*, volume 5307, pages 191–210, 2008. [page 24]

A. S. Dhillon, S. Hagan, O. Rath, and W. Kolch. MAP kinase signalling pathways in cancer. *Oncogene*, 26(22):3279–3290, 2007. [page 13]

R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4):65–74, 1988. [pages 41 and 43]

M. R. Dreher, W. Liu, C. R. Michelich, M. W. Dewhirst, F. Yuan, and A. Chilkoti. Tumor vascular permeability, accumulation, and penetration of macromolecular drug carriers. *Journal of the National Cancer Institute*, 98(5):335–344, 2006. [page 138]

P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. From CUDA to OpenCL: towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing*, 38(8):391–407, 2012. [page 27]

A. Einstein. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen. *Annalen der Physik*, 322(8):549–560, 1905, English translation see [Fürth, 1956]. [page 46]

J. Elf, A. Doncic, and M. Ehrenberg. Mesoscopic reaction-diffusion in intracellular signaling. In *SPIE*, volume 5110, pages 114–124, 2003. [page 22]

R. J. Ellis. Macromolecular crowding: an important but neglected aspect of the intracellular environment. *Current Opinion in Structural Biology*, 11:114–119, 2001. [page 22]

H. Enderling, A. R. A. Anderson, M. A. J. Chaplain, A. Beheshti, L. Hlatky, and P. Hahnfeldt. Paradoxical dependencies of tumor dormancy and progression on basic cell kinetics. *Cancer Research*, 69(22):8814–8821, 2009. [page 4]

K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 9–16, 2001. [page 42]

J. R. Faeder, M. L. Blinov, B. Goldstein, and W. S. Hlavacek. Rule-based modeling of biochemical networks. *Complexity*, 10(4):22–41, 2005. [page 60]

M. Falk and D. Weiskopf. Output-sensitive 3D line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):820–834, 2008. [page 6]

M. Falk, T. Schafhitzel, D. Weiskopf, and T. Ertl. Panorama maps with non-linear ray tracing. In *GRAPHITE 2007*, pages 9–16, 2007. [page 6]

M. Falk, M. Klann, M. Reuss, and T. Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis 2009)*, pages 169–176, 2009. [pages 5, 82, and 88]

M. Falk, S. Grottel, and T. Ertl. Interactive image-space volume visualization for dynamic particle simulations. In *Annual SIGRAD Conference*, pages 35–43, 2010a. [pages 5 and 88]

M. Falk, M. Klann, M. Reuss, and T. Ertl. 3D visualization of concentrations from stochastic agent-based signal transduction simulations. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2010)*, pages 1301–1304, 2010b. [pages 5 and 88]

M. Falk, A. Seizinger, F. Sadlo, M. Üffinger, and D. Weiskopf. Trajectory-augmented visualization of lagrangian coherent structures in unsteady flow. In *International Symposium on Flow Visualization (ISFV14)*, 2010c. [page 6]

M. Falk, M. Daub, G. Schneider, and T. Ertl. Modeling and visualization of receptor clustering on the cellular membrane. In *IEEE Symposium on Biological Data Visualization (BioVis 2011)*, pages 9–15, 2011a. [pages 5, 88, and 128]

M. Falk, M. Ott, T. Ertl, M. Klann, and H. Koeppl. Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology. In *International Conference on Computational Methods in Systems Biology (CMSB 2011)*, pages 73–82, 2011b. [page 5]

M. Falk, M. Krone, and T. Ertl. Atomistic visualization of mesoscopic whole-cell simulations. In *Eurographics Workshop on Visual Computing for Biology and Medicine (VCBM)*, volume 2, pages 123–130, 2012. [pages 5 and 88]

M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960, 1972. [page 25]

R. Fraedrich, S. Auer, and R. Westermann. Efficient high-quality volume rendering of SPH data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1533–1540, 2010. [page 123]

A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986. [page 65]

A. Fujioka, K. Terai, R. Itoh, K. Aoki, T. Nakamura, S. Kuroda, E. Nishida, and M. Matsuda. Dynamics of the Ras/ERK MAPK cascade as monitored by fluorescent probes. *Journal of Biological Chemistry*, 281(13):8917–8926, 2006. [page 55]

A. Funahashi, Y. Matsuoka, A. Jouraku, M. Morohashi, N. Kikuchi, and H. Kitano. CellDesigner 3.5: a versatile modeling tool for biochemical networks. *Proceedings of IEEE*, 96 (8):1254–1265, 2008. [pages 22, 61, and 133]

R. Fürth, editor. *Investigations on the Theory of the Brownian Movement*. Dover Publications Inc., 1956. [page 167]

D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977. [pages 18 and 21]

D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58(1):35–55, 2007. [page 21]

D. T. Gillespie and L. R. Petzold. Improved leap-size selection for accelerated stochastic simulation. *Journal of Chemical Physics*, 119(16):8229–8234, 2003. [page 22]

R. Gingold and J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices Royal Astronomical Society*, 181:375–389, 1977. [pages 22 and 34]

A. S. Glassner. Space sub-division for fast ray tracing. *IEEE Computer Graphics and Applications*, 4:15–22, 1984. [pages 35 and 65]

D. S. Goodsell. Molecule of the month. [Online]. Available: `http://www.rcsb.org/pdb/motm.do` [page 2]

D. S. Goodsell. *The Machinery of Life*. Springer, second edition, 2009. [page 143]

S. Green. Cuda particles. Technical report, NVIDIA Corporation, 2008. [page 35]

S. Grottel, G. Reina, J. Vrabec, and T. Ertl. Visual verification and analysis of cluster detection for molecular dynamics. In *IEEE Visualization 2007*, pages 1624–1631, 2007. [page 128]

S. Grottel, G. Reina, and T. Ertl. Optimized data transfer for time-dependent, GPU-based glyphs. In *IEEE Pacific Visualization Symposium (PacificVis 2009)*, pages 65–72, 2009. [page 38]

S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl. Coherent culling and shading for large molecular dynamics visualization. *Computer Graphics Forum*, 29:953–962, 2010. [pages 104 and 107]

S. Grottel, M. Krone, K. Scharnowski, and T. Ertl. Object-space ambient occlusion for molecular dynamics. In *IEEE Pacific Visualization Symposium (PacificVis 2012)*, pages 209–216, 2012. [pages 94 and 107]

G. Gruenert, B. Ibrahim, T. Lenser, M. Lohel, T. Hinze, and P. Dittrich. Rule-based spatial modeling with diffusing, geometrically constrained molecules. *BMC Bioinformatics*, 11 (1):307–320, 2010. [page 46]

S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *International Fall Workshop on Vision, Modelling and Visualization*, pages 245–252, 2003. [page 36]

K. Guo, J. Shillcock, and R. Lipowsky. Self-assembly of actin monomers into long filaments: Brownian dynamics simulations. *The Journal of Chemical Physics*, 131: 015102, 2009. [pages 46 and 59]

K. Guo, J. Shillcock, and R. Lipowsky. Treadmilling of actin filaments via brownian dynamics simulations. *Journal of Chemical Physics*, 133(15):155105, 2010. [page 59]

R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*, pages 74–93, 1990. [page 33]

M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005. [page 43]

M. Hadwiger, P. Ljung, C. R. Salama, and T. Ropinski. Advanced illumination techniques for GPU volume raycasting. In *ACM SIGGRAPH Asia 2008 courses*, 2008. [page 44]

J. H. Halton. Algorithm 247: radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964. [page 97]

E. Hammon, Jr. Practical post-process depth of field. In H. Nguyen, editor, *GPU Gems 3*, chapter 28, pages 583–606. Addison-Wesley Professional, 2007. [page 94]

G. L. Hazelbauer. Myriad molecules in motion: simulated diffusion as a new tool to study molecular movement and interaction in a living cell. *Journal of Bacteriology*, 187(1): 23–25, 2005. [page 87]

J. Heinrich, C. Vehlow, F. Battke, G. Jaeger, D. Weiskopf, and K. Nieselt. iHAT: interactive hierarchical aggregation table for genetic association data. *BMC Bioinformatics*, 13 (Suppl 8):S2, 2012. [page 146]

F. Helmchen and W. Denk. Deep tissue two-photon microscopy. *Nature Methods*, 2(12): 932–940, 2005. [page 17]

A. V. Hill. The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *Journal of Physiology*, 40(4):4–7, 1910. [page 19]

H. Hochstetter. Effiziente parallele Implementierung von hierarchischen N-Body-Algorithmen auf Multicore-Systemen mit GPU-Beschleunigung. Diploma thesis, no. 3285, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2012. [page 140]

K. C. Holmes, D. Popp, W. Gebhard, and W. Kabsch. Atomic model of the actin filament. *Nature*, 347(6288):44–49, 1990. [page 105]

C. L. Howe. Modeling the signaling endosome hypothesis: Why a drive to the nucleus is better than a (random) walk. *Theoretical Biology and Medical Modelling*, 2:43–57, 2005. [page 87]

T. Ideker, T. Galitski, and L. Hood. A new approach to decoding life: systems biology. *Annual Review of Genomics and Human Genetics*, 2(1):343–372, 2001. [page 1]

K. A. Johnson and R. S. Goody. The original Michaelis constant: translation of the 1913 Michaelis–Menten paper. *Biochemistry*, 50(39):8264–8269, 2011. [page 175]

M. F. Juette, T. J. Gould, M. D. Lessard, M. J. Mlodzianoski, B. S. Nagpure, B. T. Bennett, S. T. Hess, and J. Bewersdorf. Three-dimensional sub-100 nm resolution fluorescence microscopy of thick samples. *Nature Methods*, 5(6):527–529, 2008. [page 18]

V. Kajalin. Screen-space ambient occlusion. In W. Engel, editor, *ShaderX*^{7}, chapter 6.1, pages 413–424. Charles River Media, 2009. [page 93]

M. R. Kaplan. Space-Tracing, a constant time ray-tracer. In *ACM SIGGRAPH 1985 courses*, 1985. [pages 35 and 65]

K. Karimi, N. G. Dickson, and F. Hamze, 2010. A performance comparison of CUDA and OpenCL. [Online]. Available: http://arxiv.org/abs/1005.2581v3 [page 27]

G. Karp. *Cell and Molecular Biology: Concepts and Experiments*. John Wiley & Sons, Inc., sixth edition, 2010. [page 8]

J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival, N. Assad-Garcia, J. I. Glass, and M. W. Covert. A whole-cell computational model predicts phenotype from genotype. *Cell*, 150(2):389–401, 2012. [pages 4, 18, and 22]

M. Kass, A. Lefohn, and J. Owens. Interactive depth of field using simulated diffusion on a GPU. Technical report, Pixar Animation Studios, 2006. [page 94]

T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. *Computer Graphics (Proceedings of SIGGRAPH 1986)*, 20(4):269–278, 1986. [page 65]

J. Kessenich, D. Baldwin, and R. Rost. *The OpenGL® Shading Language*. The Khronos Group Inc., 2011, Language Version: 4.20. [page 30]

B. N. Kholodenko. Four-dimensional organization of protein kinase signaling cascades: the roles of diffusion, endocytosis and molecular motors. *Journal of Experimental Biology*, 206(12):2073–2082, 2003. [pages 14, 21, 22, 47, 82, and 87]

B. N. Kholodenko and M. R. Birtwistle. Four-dimensional dynamics of MAPK information-processing systems. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 1 (1):28–44, 2009. [page 13]

Khronos OpenCL Working Group. *The OpenCL Specification*, 2011, Version: 1.2. [page 27]

P. Kipfer, M. Segal, and R. Westermann. Uberflow: a GPU-based particle engine. In *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 115–122, 2004. [page 23]

H. Kitano. A graphical notation for biochemical networks. *BIOSILICO*, 1(5):169–176, 2003. [page 60]

H. Kitano, A. Funahashi, Y. Matsuoka, and K. Oda. Using process diagrams for the graphical representation of biological networks. *Nature Biotech*, 23(8):961–966, 2005. [page 60]

M. Klann. *Development of a Stochastic Multi-Scale Simulation Method for the Analysis of Spatiotemporal Dynamics in Cellular Transport and Signaling Processes*. Phd thesis, Universität Stuttgart, Germany, 2011. [pages 5, 24, 46, 49, 50, 63, and 141]

M. Klann and H. Koeppl. Spatial simulations in systems biology: From molecules to cells. *International Journal of Molecular Sciences*, 13(6):7798–7827, 2012. [page 133]

M. Klann, A. Lapin, and M. Reuss. Agent-based simulation of reactions in the crowded and structured intracellular environment: Influence of mobility and location of the reactants. *BMC Systems Biology*, 5(1):71, 2011. [page 54]

M. T. Klann, A. Lapin, and M. Reuss. Stochastic simulation of signal transduction: impact of the cellular architecture on diffusion. *Biophysical Journal*, 96(12):5122–5129, 2009. [pages 23, 24, 34, and 49]

T. Klein. *Exploiting Programmable Graphics Hardware for Interactive Visualization of 3D Data Fields*. Phd thesis, Universität Stuttgart, Germany, 2008. [page 38]

T. Klein and T. Ertl. Illustrating magnetic field lines using a discrete particle model. In *Vision, Modelling and Visualization (VMV 2004)*, pages 387–394, 2004. [page 36]

E. Klipp, W. Liebermeister, C. Wierling, and A. Kowald. *Systems Biology: A Textbook*. Wiley-Vch, first edition, 2009. [page 2]

K. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10(8):2703–2734, 1999. [page 60]

A. Kolb and N. Cuntz. Dynamic particle coupling for GPU-based fluid simulation. In *Symposium on Simulation Technique (ASIM)*, pages 722–727, 2005. [pages 31 and 117]

R. Kosara, S. Miksch, and H. Hauser. Semantic depth of field. In *IEEE Symposium on Information Visualization (INFOVIS 2001)*, pages 97–104, 2001. [page 94]

M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl. Interactive exploration of protein cavities. *Computer Graphics Forum*, 30(3):673–682, 2011. [pages 5 and 122]

J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2003)*, 22(3):908–916, 2003a. [page 31]

J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003*, pages 287–292, 2003b. [page 43]

J. Krüger and R. Westermann. A GPU framework for solving systems of linear equations. In M. Pharr, editor, *GPU Gems 2*, chapter 44, pages 703–718. Addison-Wesley Professional, 2005. [page 19]

P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques*, pages 451–458, 1994. [page 41]

A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. *Computer Graphics Forum*, 27(4):1235–1244, 2008. [page 106]

J. H. Lambert. *Photometria, sive de mensura et gradibus luminis, colorum et umbrae*. Eberhardt Klett, 1760. [page 89]

O. D. Lampe, I. Viola, N. Reuter, and H. Hauser. Two-level approach to efficient visualization of protein dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1616–1623, 2007. [page 104]

A. Lapin, M. Klann, and M. Reuss. Stochastic simulations of 4D-spatial temporal dynamics of signal transduction processes. In *FOSBE*, pages 421–425, 2007. [pages 24 and 46]

S. Le Grand. Broad-phase collision detection with CUDA. In H. Nguyen, editor, *GPU Gems 3*, chapter 32, pages 697–721. Addison-Wesley Professional, 2007. [page 72]

N. Le Novére et al. The systems biology graphical notation. *Nature Biotechnolgy*, 27(8): 735–741, 2009. [page 60]

S. Lee, G. Jounghyun Kim, and S. Choi. Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):453–464, 2009. [pages 31 and 97]

M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988. [pages 41 and 43]

M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3): 245–261, 1990. [page 44]

H. Li and L. Petzold. Efficient parallelization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit. *International Journal of High Performance Computing Applications*, 24(2):107–116, 2010. [pages 22 and 64]

R. Li, Y. Zheng, and D. Drubin. Regulation of cortical actin cytoskeleton assembly during polarized cell growth in budding yeast. *The Journal of Cell Biology*, 128(4):599–615, 1995. [page 59]

W. Li, K. Mueller, and A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *IEEE Visualization 2003*, pages 317–324, 2003. [pages 44 and 117]

N. Lindow, D. Baum, and H.-C. Hege. Interactive rendering of materials and biological structures on atomic and nanoscopic scale. *Computer Graphics Forum*, 31(3):1325–1334, 2012. [pages 104 and 107]

K. Lipkow, S. S. Andrews, and D. Bray. Simulated diffusion of phosphorylated CheY through the cytoplasm of Escherichia coli. *Journal of Bacteriology*, 187(1):45–53, 2005. [page 87]

H. Lodish, A. Berk, C. A. Kaiser, M. Krieger, M. P. Scott, A. Bretscher, H. Ploegh, and P. Matsudaira. *Molecular Cell Biology*. W. H. Freeman, sixth edition, 2007. [pages 7, 52, and 105]

W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques*, pages 163–169, 1987. [pages 35 and 116]

T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):1206–1213, 2006. [page 93]

N. I. Markevich, M. A. Tsyganov, J. B. Hoek, and B. N. Kholodenko. Long-range signaling by phosphoprotein waves arising from bistability in protein kinase cascades. *Molecular Systems Biology*, 2:61, 2006. [pages 82, 83, and 159]

J. Markl, editor. *Biologie*. Spektrum Akademischer Verlag GmbH, second edition, 2000, German translation of [Campbell, 1996]. [page 7]

T. T. Marquez-Lago and K. Burrage. Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics. *Journal of Chemical Physics*, 127(10): 104101–104101–9, 2007. [page 22]

G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003. [page 70]

W. Mather, N. Cookson, J. Hasty, L. Tsimring, and R. Williams. Correlation resonance generated by coupled enzymatic processing. *Biophysical Journal*, 99(10):3172–3181, 2010. [page 83]

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. [page 70]

N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. [page 40]

J. A. McCammon, B. R. Gelin, and M. Karplus. Dynamics of folded proteins. *Nature*, 267 (5612):585–590, 1977. [pages 22 and 34]

O. Medalia, I. Weber, A. S. Frangakis, D. Nicastro, G. Gerisch, and W. Baumeister. Macromolecular architecture in eukaryotic cells visualized by cryoelectron tomography. *Science*, 298(5596):1209–1213, 2002. [page 16]

L. Michaelis and M. L. Menten. Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, 49:333–369, 1913, English translation see [Johnson and Goody, 2011]. [pages 19, 55, and 156]

A. Mogilner, J. Allard, and R. Wollman. Cell polarity: Quantitative modeling as a tool in cell biology. *Science*, 336(6078):175–179, 2012. [pages 58 and 59]

K. Mosaliganti, L. Cooper, R. Sharp, R. Machiraju, G. Leone, K. Huang, and J. Saltz. Reconstruction of cellular biological structures from optical microscopy data. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):863–876, 2008. [page 151]

J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15(1):119–129, 1999. [page 34]

C. Müller, S. Grottel, and T. Ertl. Image-space GPU metaballs for time-dependent particle data sets. In *Vision, Modelling and Visualization (VMV 2007)*, pages 31–40, 2007. [pages 35 and 123]

R. F. Murphy. Cellorganizer: image-derived models of subcellular organization and protein distribution. *Computational Methods in Cell Biology*, 110:179–193, 2012. [page 151]

NVIDIA Corporation. *CUDA C Programming Guide*, 2011, Version: 4.1. [pages 27 and 32]

M. Oren and S. Nayar. Generalization of Lambert's reflectance model. In *SIGGRAPH 1994*, pages 239–246, 1994. [page 89]

J. Pahle. Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. *Briefings in Bioinformatics*, 10(1):53–64, 2009. [page 18]

B. Palsson. The challenges of in silico biology. *Nature Biotechnolgy*, 18(11):1147–1150, 2000. [page 1]

A. Partikian, B. Ölveczky, R. Swaminathan, Y. Li, and A. Verkman. Rapid diffusion of green fluorescent protein in the mitochondrial matrix. *The Journal of Cell Biology*, 140 (4):821–829, 1998. [page 49]

M. Pharr, editor. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Add, 2005. [page 31]

M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, second edition, 2010. [page 92]

B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975. [pages 44 and 89]

I. Pirson, N. Fortemaison, C. Jacobs, S. Dremier, J. E. Dumont, and C. Maenhaut. The visual display of regulatory information and networks. *Trends in Cell Biology*, 10(10): 404–408, 2000. [page 60]

S. J. Plimpton and A. Slepoy. Microbial cell modeling via reacting diffusive particles. *Journal of Physics: Conference Series*, 16:305–309, 2005. [page 24]

M. Pogson, R. Smallwood, E. Qwarnstrom, and M. Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85(1):37–45, 2006. [pages 24 and 54]

S.-D. Poisson. *Recherches sur la probabilité des jugements en matière criminelle et en matière civile: précédées des règles générales du calcul des probabilités*. Bachelier, 1837. [page 97]

T. Porter and T. Duff. Compositing digital images. *Computer Graphics (Proceedings of SIGGRAPH 1984)*, 18(3):253–259, 1984. [page 41]

M. Potmesil and I. Chakravarty. A lens and aperture camera model for synthetic image generation. *Computer Graphics (Proceedings of SIGGRAPH 1981)*, 15(3):297–305, 1981. [pages 94 and 95]

G. Reina and T. Ertl. Hardware-accelerated glyphs for mono-and dipoles in molecular dynamics visualization. In *Eurographics/IEEE VGTC Symposium on Visualization*, pages 177–182, 2005. [page 38]

S. Rialle, L. Felicori, C. Dias-Lopes, S. Pérès, S. El Atia, A. R. Thierry, P. Amar, and F. Molina. BioNetCAD: design, simulation and experimental validation of synthetic biochemical networks. *Bioinformatics*, 26(18):2298–2304, 2010. [page 133]

F. M. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977. [page 121]

T. Rindfleisch. Photometric method for lunar topography. *Photogrammetric Engineering*, 32(2):262–276, 1966. [page 89]

T. Ritschel, T. Grosch, and H.-P. Seidel. Approximating dynamic global illumination in image space. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2009*, 2009. [pages 31 and 93]

E. Roberts, J. E. Stone, L. Sepúlveda, W.-M. W. Hwu, and Z. Luthey-Schulten. Long time-scale simulations of in vivo diffusion using GPU hardware. In *International Parallel and Distributed Processing Symposium*, pages 1–8, 2009. [page 64]

J. Roberts. State of the art: coordinated & multiple views in exploratory visualization. In *International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*, pages 61–71, 2007. [page 135]

S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Eurographics/IEEE TCVG Symposium on Visualization (VisSym 2003)*, pages 231–238, 2003.

K. Roettiger, J. Burns, and C. Loken. When clusters collide — a numerical hydro/n-body simulation of merging galaxy clusters. *Astrophysical Journal*, 407(2):L53–L56, 1993. [page 139]

P. Sabella. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics (Proceedings of SIGGRAPH 1988)*, 22(4):51–58, 1988. [pages 39, 41, and 43]

T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):197–206, 1990. [pages 31, 92, and 107]

T. Schafhitzel, M. Falk, and T. Ertl. Real-time rendering of planets with atmospheres. *Journal of WSCG*, 15:91–98, 2007. [page 6]

H. Schechter and R. Bridson. Ghost SPH for animating water. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, 31(4):61, 2012. [page 22]

T. Scheuermann and N. Tatarchuk. Improved depth of field rendering. In W. Engel, editor, *ShaderX*^{3}, chapter 4.4, pages 363–377. Charles River Media, 2004. [pages 31, 96, 97, and 98]

R. Schmidt, C. A. Wurm, A. Punge, A. Egner, S. Jakobs, and S. W. Hell. Mitochondrial cristae revealed with focused light. *Nano Letters*, 9(6):2508–2510, 2009. [page 17]

J. P. Schulze, M. Kraus, U. Lang, and T. Ertl. Integrating pre-integration into the shear-warp algorithm. In *International Workshop on Volume Graphics*, pages 109–118, 2003.

M. Segal and K. Akeley. *The OpenGL® Graphics System: A Specification*. The Khronos Group Inc., 2011, Version: 4.2 (Core Profile). [pages 27, 28, and 30]

L. Segel and M. Slemrod. The quasi-steady-state assumption: a case study in perturbation. *SIAM review*, 31(3):446–477, 1989. [pages 55 and 83]

L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3):1–15, 2008. [page 25]

J. C. Shillcock. Insight or illusion? Seeing inside the cell with mesoscopic simulations. *HFSP Journal*, 2(1):1–6, 2008. [page 2]

P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):63–70, 1990. [page 41]

D. Shreiner and the Khronos OpenGL ARB Working Group. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1.* Addison-Wesley, seventh edition, 2009. [page 30]

C. T. Silva, J. Freire, and S. P. Callahan. Provenance for visualizations: reproducibility and beyond. *Computing in Science and Engineering*, 9(5):82–89, 2007. [page 150]

B. D. Slaughter, S. E. Smith, and R. Li. Symmetry breaking in the life cycle of the budding yeast. *Cold Spring Harbor Perspectives in Biology*, 1(3):a003384, 2009. [pages 57 and 59]

A. Smith, W. Xu, Y. Sun, J. Faeder, and G. Marai. RuleBender: integrated visualization for biochemical rule-based modeling. In *IEEE Symposium on Biological Data Visualization (BioVis 2011)*, pages 103–110, 2011. [page 60]

W. Smith. Stochastic models for an enzyme reaction in an open linear system. *Bulletin of Mathematical Biology*, 33(1):97–115, 1971. [page 55]

M. W. Sneddon and T. Emonet. Modeling cellular signaling: taking space into the computation. *Nature Methods*, 9(3):239–241, 2012. [page 19]

V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435(7042):629–636, 2005. [page 139]

S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In *Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 187–195, 2005. [pages 43 and 117]

J. R. Stiles and T. M. Bartol. Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. In E. D. Schutter, editor, *Computational Neuroscience: Realistic Modeling for Experimentalists*, chapter 5, pages 87–127. CRC Press, 2001. [page 133]

J. R. Stiles, D. Van Helden, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Miniature end-plate current rise times less than 100 microseconds from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proceedings of the National Academy of Sciences*, 93(12):5747–5752, 1996. [page 24]

A. B. Stundzia and C. J. Lumsden. Stochastic simulation of coupled reaction-diffusion processes. *Journal of Computational Physics*, 127(1):196–207, 1996. [page 22]

P. S. Swain, M. B. Elowitz, and E. D. Siggia. Intrinsic and extrinsic contributions to stochasticity in gene expression. *Proceedings of the National Academy of Sciences*, 99 (20):12795–12800, 2002. [page 149]

K. Takahashi, S. N. V. Arjunan, and M. Tomita. Space in systems biology of signaling pathways – towards intracellular molecular crowding in silico. *FEBS Letters*, 579(8): 1783–1788, 2005. [pages 18 and 19]

M. Tarini, P. Cignoni, and C. Montani. Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006. [pages 92 and 100]

D. P. Tolle and N. Le Novère. Particle-based stochastic simulation in systems biology. *Current Bioinformatics*, 1(3):315–320, 2006. [pages 18 and 24]

S. Trinh, P. Arce, and B. R. Locke. Effective diffusivities of point-like molecules in isotropic porous media by Monte Carlo simulation. *Transport in Porous Media*, 38(3):241–259, 2000. [page 49]

T. E. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 28(3):165–178, 2004. [page 18]

K. van Kooten, G. van den Bergen, and A. Telea. Point-based visualization of metaballs on a GPU. In H. Nguyen, editor, *GPU Gems 3*, chapter 7, pages 123–148. Addison-Wesley Professional, 2007. [page 35]

J. A. van Meel, A. Arnold, D. Frenkel, S. F. P. Zwart, and R. G. Belleman. Harvesting graphics power for MD simulations. *Molecular Simulation*, 34(3):259–266, 2007. [page 23]

J. S. van Zon and P. R. Ten Wolde. Green's-function reaction dynamics: a particle-based approach for simulating biochemical networks in time and space. *The Journal of Chemical Physics*, 123:234910, 2005. [page 23]

M. von Smoluchowski. Versuch einer mathematischen Theorie der Koagulationskinetik kolloider Lösungen. *Zeitschrift für Physikalische Chemie*, 92:129–168, 1917. [page 24]

P. Waage and C. M. Guldberg. Studies concerning affinity. *Forhandlinger: Videnskabs-Selskabet i Christiana*, 12:35–45, 1864, English translation see [Waage and Guldberg, 1986]. [pages 19 and 50]

P. Waage and C. M. Guldberg. Studies concerning affinity. *Journal of Chemical Education*, 63(12):1044–1047, 1986, translated by H. I. Abrash. [page 179]

S. Waldherr, J. Hasenauer, and F. Allgöwer. Estimation of biochemical network parameter distributions in cell populations. In *IFAC Symposium on System Identification*, pages 1265–1270, 2009. [page 19]

C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers Inc., 2004. [page 2]

J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Blöschl, and E. Gröller. World lines. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1458–1467, 2010. [page 150]

B. Weber, G. Greenan, S. Prohaska, D. Baum, H.-C. Hege, T. Müller-Reichert, A. A. Hyman, and J.-M. Verbavatz. Automated tracing of microtubules in electron tomograms of plastic embedded samples of *Caenorhabditis elegans* embryos. *Journal of Structural Biology*, 178(2):129–138, 2012. [page 151]

J. R. Weimar and J. Boon. Class of cellular automata for reaction-diffusion systems. *Physical Review E*, 49(2):1749–1752, 1994. [page 19]

M. Weisel, E. Proschak, and G. Schneider. PocketPicker: analysis of ligand binding-sites with shape descriptors. *Chemistry Central Journal*, 1(7):1–17, 2007. [page 121]

D. Weiskopf. *GPU-based Interactive Visualization Techniques*. Springer, Berlin, 2007. [page 33]

D. Weiskopf and T. Ertl. Real-time depth-cueing beyond fogging. *Journal of Graphical Tools*, 7(4):83–90, 2002. [page 91]

R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques*, pages 169–177, 1998. [page 41]

L. Westover. Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1990)*, 24(4):367–376, 1990. [page 41]

D. Xue, C. Zhang, and R. Crawfis. iSBVR: isosurface-aided hardware acceleration techniques for slice-based volume rendering. In *Eurographics/IEEE VGTC Workshop on Volume Graphics*, pages 207–215, 2005. [page 41]

S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. In *Eurographics Workshop on Rendering*, pages 45–56, 1998. [page 94]