Johann Wolfgang Goethe-Universität
Frankfurt am Main

Fachbereich Biologie und Informatik (15)
Institut für Informatik
Lehrstuhl für Graphische Datenverarbeitung

in corporation with

KECK GRADUATE INSTITUTE
for applied Life Sciences

Diplomarbeit

vorgelegt von: Frank Bergmann
Matr. Nr.: 17 28 365
E-Mail: fbergman@kgi.edu

Betreuer: Dipl.-Wirtsch.-Inform. Daniel F. Abawi
Dipl.-Biol. Jens Barthelmes

September 2004

Erstprüfer: Prof. Dr.-Ing. Detlef Krömker
Zweitprüfer: Prof. Dr. Oswald Drobnik
Drittprüfer: Prof. Dr. Georg Schnitger

Configuration, simulation and visualization of simple biochemical reaction-diffusion systems in 3D

Zusammenfassung

Reaktionsdiffusionssysteme wurden in der Vergangenheit immer wieder mit großem Interesse untersucht. Dabei betrachtete man meist die folgenden zwei Themengebiete: Einerseits Reaktionsdiffusionssysteme und ihre Rolle bei der Morphogenese, der Entwicklung von lebenden Zellen und andererseits Reaktionsdiffusionssysteme im Hinblick auf Muster, die durch sie begründet werden. Zu diesen gehören einfache Punkt- und Streifenmuster bis hin zu komplexen Mustern, wie man sie an Tierfellen sehen kann. Bei diesen Untersuchungen beschränkte man sich meist auf die ein- oder zweidimensionalen Varianten dieser Systeme.

Ziel dieser Arbeit ist das Entwerfen einer Anwendung zum Konfigurieren, Simulieren und Visualisieren von einfachen dreidimensionalen Reaktionsdiffusionssystemen. Eine solche Anwendung ist hilfreich zur Untersuchung von räumlichen Effekten in metabolischen Prozessen. Weiterhin ist zu hoffen, durch sie die Entstehung von dreidimensionalen Mustern verfolgen und erklären zu können.

Um dieses Ziel zu erreichen, wurde die folgende Vorgangsweise gewählt. Zunächst wurde untersucht welche Vorarbeit auf den einzelnen Teilgebieten schon geleistet wurde. Die Konfiguration des dreidimensionalen Reaktionsvolumens gleicht dabei am ehesten einer Modellierungsapplikation, denn ein Großteil der Konfigurierungsarbeit ist das Platzieren von Elementen (Molekühlkonzentrationen, Kanälen oder Membranen) in das Reaktionsvolumen. Um geeignete Methoden zur Eingabe zu finden, wurden gebräuchliche Modellierungsapplikationen untersucht, um herauszufinden, wie diese die Aufgabe lösen. Ein weiterer Teil der Analyse beschäftigte sich dann mit der Suche nach einem geeigneten Dateiformat für die Speicherung der Konfigurationsdaten. Aufgrund des Zieles, ein existierendes Dateiformat zu verwenden, um unter Umständen auch Anwendungen von Drittanbietern nutzen zu können, kamen die XML Dialekte CellML und SBML in Frage. Durch die weite Verbreitung von SBML und eine nachvollziehbarere Struktur wurde letztendlich SBML gewählt. Daraufhin wurde analysiert, welche Reaktionsdiffusionssysteme in der Vergangenheit untersucht worden sind. Ein integraler Bestandteil des Simulationskerns ist eine Integriereinheit. Daher beschäftigte sich die Analyse auch mit verschiedenen Methoden zum Integrieren. Schließlich wurden häufig genutzte Volumenvisualisierungsalgorithmen und –bibliotheken untersucht.

Die durch die Analyse gewonnenen Vorüberlegungen wurden dann in einer Konzeptionsphase weitergeführt. Als Kernpunkt stand hier die Erweiterbarkeit im Vordergrund. Da sowohl die zu untersuchenden Reaktionsdiffusionssysteme einfach abgeändert werden sollten, als auch die unterschiedliche Darstellung der Simulationsdaten unterstützt werden sollte, wurde ein Pluginsystem konzipiert. Weiterhin wurde in dieser Phase beschlossen, das Gesamtprojekt in die Teile Konfiguration, Simulation und Visualisierung zu unterteilen. Die Idee hierbei war, der logischen Struktur zu folgen, im ersten Schritt eine Konfiguration zu erstellen, diese in einem zweiten Schritt zu simulieren und schließlich in einem dritten Schritt die Simulationsdaten anzuzeigen.

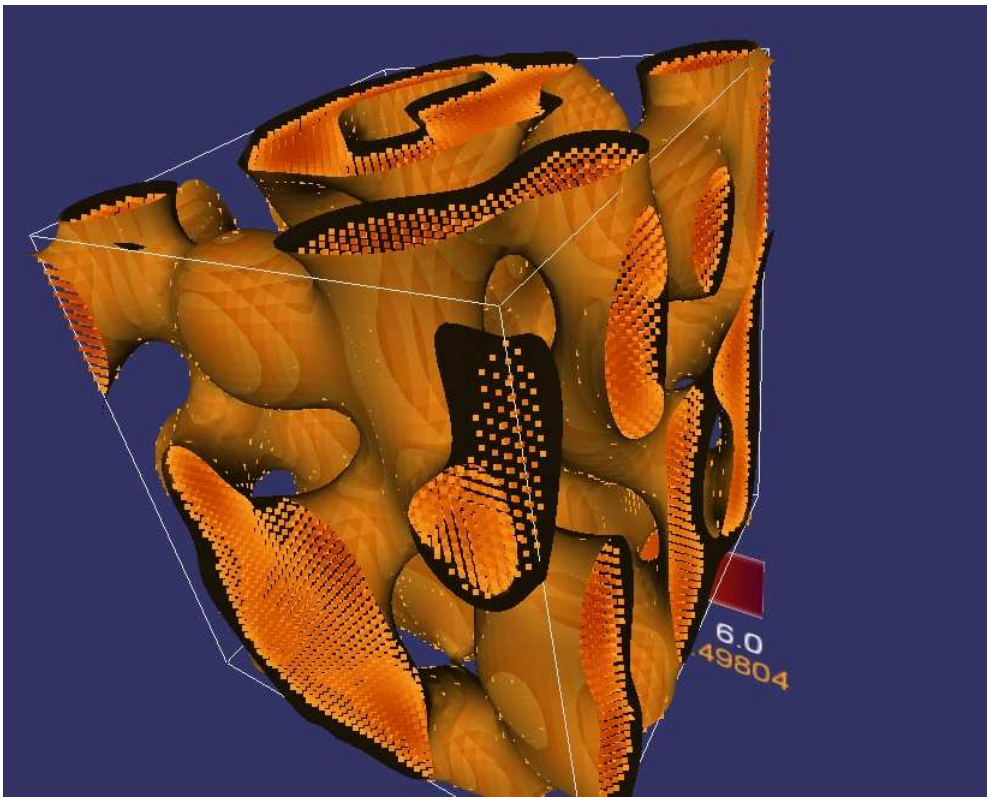


Figure 1: Visualisierung einer Brusselator Simulation (50x50x50)

Auf der Basis von C++, erweitert um das Windowing Toolkit Qt und die Scenengraph-API OpenSceneGraph, ist daraufhin eine prototypische Implementierung in drei Teilen entstanden. Für die Konfiguration wurde die Anwendung „configuration unit“ erstellt. Diese Applikation erlaubt das Festlegen aller Parameter, die für die Simulation benötigt werden. Dazu gehören das Festlegen der Größe des Reaktionsvolumens, das Auswählen des zu simulierenden Reaktionsdiffusionssystems und das Einfügen von verschiedenen Elementen (Molekühlkonzentrationen, Kanälen und Membranen) in das Reaktionsvolumen. Aufgrund des ausgewählten Dateiformates (SBML level 2) ist es möglich, diese Konfiguration mit dem Programm JDesigner nachträglich noch zu erweitern. JDesigner erlaubt das visuelle Festlegen von Reaktionsnetzwerken.

Das zweite Programm, dass umgesetzt wurde, ist die Anwendung „simulation unit“. Dieses berechnet aufgrund der Konfiguration solange neue Iterationen, bis der Benutzer das Berechnen abbricht. Dabei stehen in der prototypischen Form drei Simulationsmodi zur Verfügung: zum einen das Simulieren einfacher Diffusion, zum anderen das Simulieren des Brusselator Reaktionsdiffusionssystem (ein einfaches Reaktionsdiffusionssystem bestehend aus Diffusion und vier Reaktionsgleichungen) und schließlich ein Simulationsmodus, der die mittels JDesigner veränderten Konfigurationen durch die Systems Biology Workbench (SBW) an das Programm Jarnac zum Berechnen weiterleitet.

Die letzte der drei implementierten Anwendungen (“visualization unit“) hat die Aufgabe der Visualisierung der Simulationsergebnisse. Dafür stehen zwei Ausführungsmodi zur Verfügung. Zum einen ist es möglich, mit diesem Programm die Simulationsergebnisse aus einem vorherigen Lauf des Simulationsprogramms wiederzugeben (dieser Modus wird in dieser Arbeit auch „offline“ Visualisierung genannt, da die Simulation von der Visualisierung entkoppelt ist). Zum anderen ist es auch möglich, die Visualisierung im „online“ Modus auszuführen. In diesem Fall wird eine Konfiguration verwendet, um damit Simulation und Visualisierung alternierend auszuführen. Beide Ausführungsmodi haben ihre Vor- und Nachteile. Die „Offline“ Visualisierung ist recht schnell,

benötigt allerdings recht viel verfügbaren Speicherplatz auf der Festplatte. Dadurch, dass die einzelnen Iterationen auf Festplatte abgespeichert werden, ist es allerdings auch möglich, diese öfters und in beliebiger Reihenfolge wiederzugeben. Beim „online“ Modus werden im Allgemeinen keine Daten auf die Festplatte geschrieben, demnach ist es nicht möglich zu einer vorherigen Iteration zurückzukehren (es sei denn man beginnt erneut mit der Simulation durch Neuinitialisieren mit der ursprünglichen Konfiguration). Beiden Ausführungsmodi gemein ist, dass der Benutzer die Art der Visualisierung frei wählen kann. Darüber hinaus können die gewünschten Informationen noch genauer spezifiziert werden. Durch die Definition von Clipvolumen (diese erlauben ein „Zurechtschneiden“ der Anzeige, so dass auch innere Strukturen angezeigt werden können) kann die Anzeige vereinfacht werden und bestimmte Gebiete genauer beobachtet werden. Weiterhin ermöglicht ein Ändern des Schwellenwertes, für den zum Beispiel die Oberflächen generiert werden, das Hervorheben anderer Strukturen im Reaktionsvolumen. Normalerweise wird dieser Schwellenwert automatisch durch den Simulationskern „geschätzt“, basierend auf dem Minimum-, Maximum- und Mittelwert der aktuellen Iteration. Die Abbildung auf Seite 71 demonstriert wie die Veränderung des Schwellenwertes die Ausgabe beeinflusst.

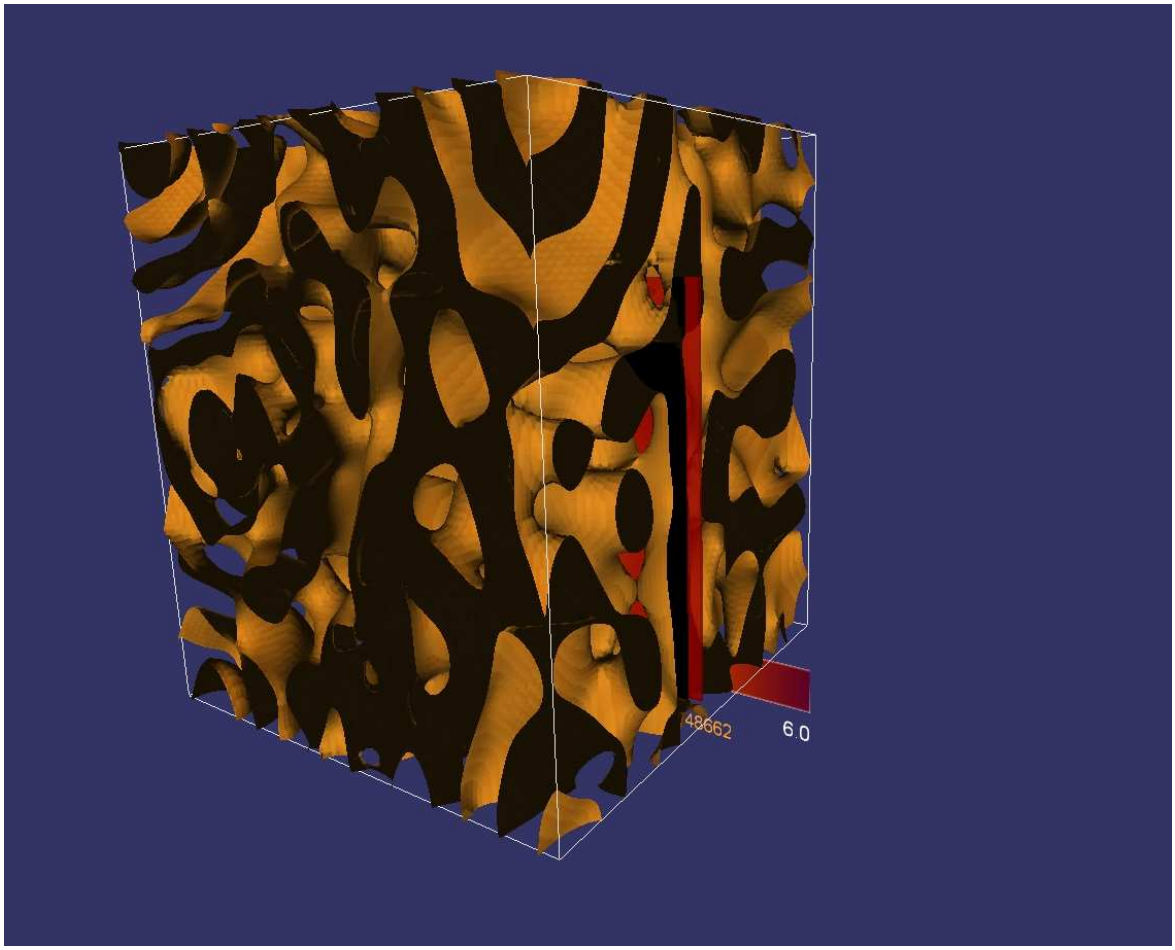


Figure 2: Visualisierung einer Brusselator Simulation (100x100x100)

Die Abbildungen dieses Abschnittes demonstrieren die Visualisierungsmöglichkeiten der Visualisierungsapplikation.

Schlüsselwörter: Reaktions-Diffusions-Systeme, Simulation, Volumenvisualisierung

Abstract

Reaction-diffusion systems have been widely studied through the years. They are suspected of being a vital part in the creation of life. Also reaction-diffusion processes give rise to the formation of complex patterns ranging from spots to stripes to labyrinth like constructs. Since skin patterns on animals (like seen on cheetahs, zebras or tropical fish) can be recreated using reaction-diffusion simulations, the formation of these skin patterns is alleged to reaction-diffusion processes.

The research of reaction-diffusion systems was mostly restricted to their one- and two-dimensional variants. The aim of this thesis is to extend that research to the three-dimensional case. Therefore the creation of a program suite for the configuration, simulation and visualization of three-dimensional reaction-diffusion systems will be described. In order to support a wide range of reaction-diffusion systems a plug-in system will be described that allows providing new reaction-equations. Employing the Systems Biology Workbench it will also be possible to visually design reaction networks and simulating them. A similar plug-in system will be used for the visualization of three-dimensional reaction-diffusion simulations. This will allow extending the existing visualization modes in the near future. The prototypic implementation of the visualization will contain standard volume-visualization algorithms presenting the simulation data through voxel-sets, generated iso-surfaces or textures.

Keywords: reaction-diffusion systems, simulation, volume-visualization

Ehrenwörtliche Erklärung

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Frankfurt am Main, 22. September 2004

Frank Bergmann

Table of Contents

Preface

Acknowledgements

In this section the author likes to express his thanks to everyone extending their support towards creating this thesis.

This thesis wouldn't have been possible without the strong support of Prof. Dr.-Ing. Detlef Kroemker of the University of Frankfurt and Prof. Herbert Sauro of the Keck Graduate Institute. Thanks go out to my advisors Dipl.-Wirtsch.-Inform. Daniel F. Abawi and Dipl.-Biol. Jens Barthelmes for guiding me through the process of writing this thesis and formulating the problem. The author is deeply grateful to Prof. Dr. Oswald Drobnik and Prof. Dr. Georg Schnitger for their willingness to review this thesis.

Many thanks also to Alpan Raval, Vijay Chikarmane, Cameron Wellock, Sri Rama Krishna Paladugu, Anastasia Deckard and Abhishek Agrawal for many interesting discussions around this project. These people made the authors stay at the KGI a joy. The author also likes to thank Christoph Karwoth, Martin Klossek and Matthias Pfeiffer for many critical comments and their willingness to test the created application.

Finally the author likes to express his deep thanks towards his parents for their support, understanding and encouragement they have provided throughout his education.

Programs used

This document was created using Microsoft Word 2003[1], MathType 5.2[2], Microsoft PowerPoint 2003 and Corel PHOTO-PAINT[3] 11. Finally Adobe Acrobat 6.0[4] was used in order to create the PDF file.

The thesis, some more pictures as well as source code will be available under:

http://public.kgi.edu/~fbergman/thesis/results.html

and on the attached CD-ROM. I will be happy about comments and critic. You can reach me via e-mail under fbergman@kgi.edu .

[1] Microsoft Word 2003 and Microsoft PowerPoint 2003 are trademarks of the Microsoft Corporation
[2] MathType 5.2 is a trademark of Design Science, Inc.
[3] Corel PHOTO-PAINT 11 is a trademark of the Corel Corporation
[4] Adobe Acrobat 6.0 is a trademark of Adobe Systems Incorporated

1 Introduction

1.1 Motivation

Reaction-diffusion systems have been studied since the invention of the first digital computers. In the 1950s Alan Turing proposed that reaction-diffusion processes might be responsible for morphogenesis (i.e. the evolutionary development of the structure of an organism or part of an organism). He and many other researchers gave reaction-diffusion systems and the possibility of creating patterns using reaction-diffusion systems quite some thought. Most of the work in this area has been concerned with two-dimensional systems.

The range of patterns attributed to reaction-diffusion systems range from spots, stripes, spirals to labyrinth-like constructs to patterns as seen on animal skins. This thesis will try and show how these patterns will behave in the three-dimensional case. Possible benefits from this work might be information about the way these patterns form.

The mathematical principles will probably extend rather easily from the one- and two-dimensional case. Challenges will probably arise through the volume-visualization.

1.2 Problem

This section will state the problem that is to be solved in this thesis. The process of formulating this problem was done in close cooperation with the authors advisors Dipl.-Wirtsch.-Inform. Daniel F. Abawi and Dipl.-Biol. Jens Barthelmes. It was also acknowledged by Prof. Herbert Sauro of the Keck Graduate Institute.

The title for this thesis will be:

> *Three-Dimensional Configuration, Simulation and Visualization of Simple Biochemical Reaction-Diffusion Systems*

The thesis is to be placed in the following biological background:

> *In biological systems, molecules of different species diffuse within the reaction compartments and interact with each other, ultimately giving rise to such complex structures like living cells. In order to investigate the formation of sub cellular structures and patterns (e.g. signal transduction) or spatial effects in metabolic processes, it would be helpful to use simulations of such reaction-diffusion systems. Pattern formation has been extensively studied in two dimensions. However, the extension to three-dimensional reaction-diffusion systems poses some challenges to the visualization of the processes being simulated.*

This background is met by the following definition for the scope of this thesis: The aim of this thesis is the specification and development of algorithms and methods for the three-dimensional configuration, simulation and visualization of biochemical reaction-diffusion systems consisting of a small number of molecules and reactions. After an initial review of existing literature about 2D/3D reaction-diffusion systems, a 3D simulation algorithm (PDE solver), based on an existing 2D-simulation algorithm for reaction-diffusion systems written by Prof. Herbert Sauro, has to be developed. In a succeeding step, this algorithm has to be optimized for high performance. A proto-

typic 3D configuration tool for the initial state of the system has to be developed. This basic tool should enable the user to define and store the location of molecules, membranes and channels within the reaction space of user-defined size. A suitable data structure has to be defined for the representation of the reaction space. The main focus of this thesis is the specification and prototypic implementation of a suitable reaction space visualization component for the display of the simulation results. In particular, the possibility of 3D visualization during course of the simulation has to be investigated. During the development phase, the quality and usability of the visualizations has to be evaluated in user tests. The simulation, configuration and visualization prototypes should be compliant with the Systems Biology Workbench to ensure compatibility with software from other authors. The thesis is carried out in close cooperation with Prof. Herbert Sauro at the Keck Graduate Institute, Claremont, CA, USA. Due to this international cooperation the thesis will be written in English.

1.3 Structure of this thesis

This section will detail how the chapters will be structured. The last part will then take a glimpse into the threefold-ness of the project.

The first chapter is meant as an introductory chapter. It begins with a motivational part, continues with stating the problem of the project and finally ends in this explanation of the structure of this thesis. Chapter two will explain the main terms for the thesis. It will shortly give explanations / definitions for reaction-diffusion, volume-visualization and the Systems Biology Workbench. With these definitions it will be easy to follow what this thesis is about. In chapter three there will be the analysis of work previously done in this field. Since three-dimensional reaction-diffusion simulators are hard to find the chapter will be divided into a part for the configuration, the simulation and finally the visualization of three-dimensional reaction-diffusion systems. Chapter four then will contain the conception of the thesis. It will, again be separated into the three parts (configuration, simulation and visualization), detail the concepts for realizing the project. The fifth chapter will talk about the implementation side of this project. It will explain the architecture of the software system. Also an idea for an optimization of the simulation-core will be given. There will be a section on problems that were encountered and solved. The chapter ends with a section on limitations of the system. In Chapter six an evaluation can be found. It will ascertain whether the introduced concepts were the right concept to meet the problem defined here. After that chapter seven will contain a final summary of the project along with an outlook of how the project might be continued in the next future. Finally an appendix section was added that contains more information about the included program CD, some examples, the schema definitions used and the bibliography.

Perhaps here is the right place to introduce the idea of the threefold-ness of this project. In order to obtain three-dimensional visualizations of reaction-diffusion systems the following tasks have to be done. First of all, before a simulation can take place, a configuration has to be created that details how the simulation is supposed to be running and what it should simulate. Secondly, before any data can be visualized there have to be some sort of simulation results. This leads ultimately to the three parts: configuration, simulation and visualization. The whole project then exists though following these three parts beginning with configuration, and then simulation alternated with visualization.

2 Terminology

This first section will introduce the central terms for this project. These explanations will by no means be exhaustive and so references will be given and it will be referred to later chapters. First there will be a short description of the reaction-diffusion process. Next then there will be a description and categorization of volume-visualization. And as a last major point in this section the System Biology Workbench (SBW), and the file format SBML and its role in SBW will be introduced.

2.1 Reaction-Diffusion-Systems

Reaction-diffusion systems have been widely studied throughout the years. Since the aim of this thesis is the configuration, simulation and visualization of three-dimensional reaction-diffusion systems the first thing to do is to define what a reaction-diffusion system is.

As defined in [Mur01] wherever there is an assemblage of elements (Here elements could stand for cells, bacteria, chemicals …) each element will move around in a random way (also known as Brownian motion). As a result of the elements random motion the elements spread out. If this individual random motion results in a regular motion of the group that regular motion is called diffusion. Though there might be interaction between the elements in which case the overall movement is not simple diffusion. It is very hard to derive the overall movement from knowledge of the individual elements movements. Instead a model of the global behavior is derived by looking at element density or concentration.

In other words reaction-diffusion is a process where two or more elements diffuse over a surface (or volume in the three-dimensional case) and react with one another to produce stable patterns. Thereby a variety of spot and stripe patterns can be produced. In nature these patterns can be seen on skin patterns of various animals like zebras, leopards and such. This definition was derived from [Tur92].

Since reaction-diffusion systems pose the starting point for many researchers there is no shortage of references. The author found [Mur01], [Tur52], [Tur91] and [Har93] most insightful. In this thesis reaction-diffusion systems will be further analyzed in section 3.3.1 and 4.3.1.

2.2 Volume-Visualization

The visualization of data preserving three-dimensional spatial structures is called *volume-visualization*. Here images are created from scalar or vector datasets defined on higher dimensional grids i.e. a multidimensional dataset is projected onto the image plane. A good general introduction can be found in [Fol90], [Enc97] and [Owe99]. Volume-visualization is commonly used in medical environments (computer-aided tomography (CT), nuclear magnetic resonance (NMR), single photon emission computed tomography (SPECT)) or in the visualization of meteorological data. Volume-visualization is still a task that requires tremendous resources from the graphics hardware. Thus specialized hardware or even clusters are employed. Due to the rapid development of hardware in the PC sector slowly these two areas begin to merge. See also [Eng00].

In the case of visualizing three-dimensional reaction-diffusion simulations the demand on the hardware is an order higher than for regular visualization of volume data. The reason for this is, that each iteration of the simulation will result in a new dataset. Thus the kind of volume data present is animated volume data. [Gut01] presents a way to use three-dimensional wavelet transforms and motion compensation techniques to achieve interactive frame-rates visualizing animated volume data on a single PC.

Volume-visualization algorithms are mainly divided into direct-volume-rendering, interactive methods and surface-fitting algorithms. For an analysis of commonly used algorithms see section 3.4.2 and [Elv92].

2.3 Systems Biology Workbench (SBW)

The Systems Biology Workbench (SBW) was developed with the goal to create a software framework that allows arbitrary components to communicate with each other. The components should not be restricted to any particular programming language or operating system.

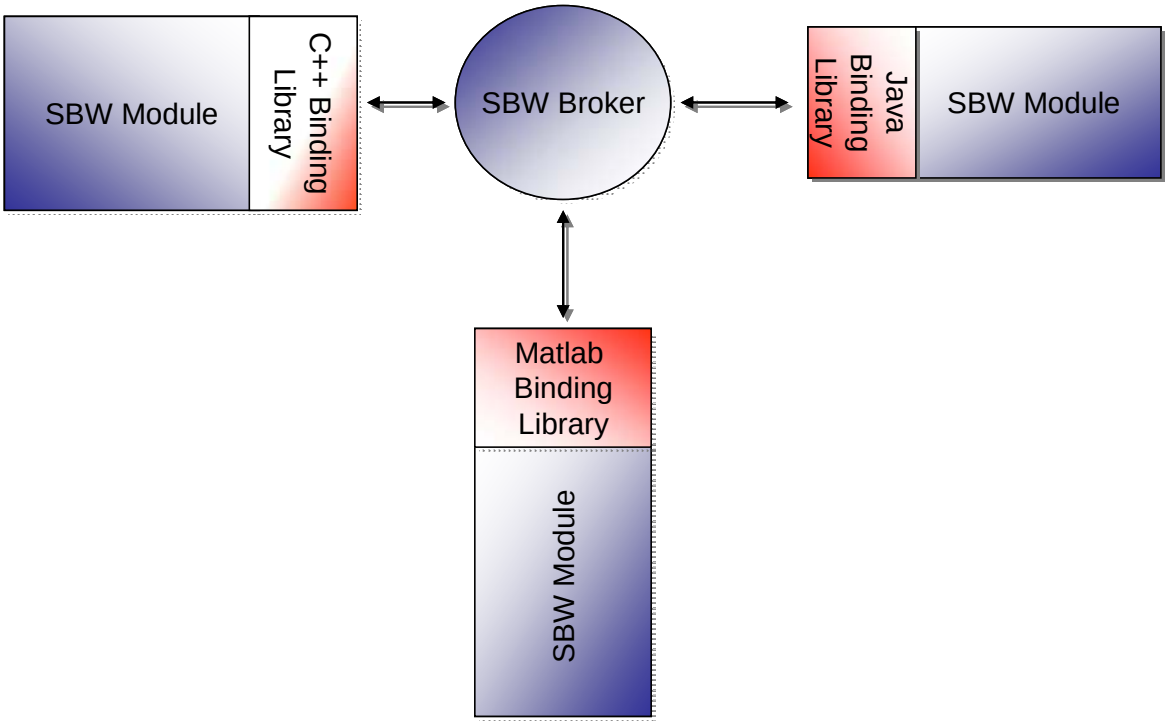


Figure 3: SBW message flow

"Figure 3: SBW message flow" displays the normal message flow through the SBW Framework. The core of SBW is the SBW broker. As soon as the first SBW module is started on a client machine the broker is started along with it. After that the full functionality of all registered / running modules is available to the started module. Access to remote modules is also possible.

Enabling a component for use in the SBW framework is fairly easy done with minimal changes to the existing program code. This is possible through binding libraries which exist for a variety of programming languages including C, C++, Delphi, JAVA, Python, Perl and Matlab. Once the application component is SBW-enabled it has instant access to a variety of new functionality.

Available SBW-modules include simulation-, modeling- and optimization-modules. These can be easily accessed via the binding libraries. (They can be queried either by name or by category.). A full list of currently available modules can be found at: http://www.sbw-sbml.org/.

SBW-modules have adopted the Systems Biology Markup Language (SBML) as their native file format for saving model information. An example for the seamless integration of features from one application into another application using SBML can be seen on the example of JDesigner and Jarnac. “JDesigner is a visual tool for the layout of reaction networks, including metabolic, signal transduction and gene regular networks. …” (JDesigner Help file version 1.91c). Jarnac is a programming language for numerical analysis. It has special capabilities for analyzing biological models which are usually entered in script form. An alternative is to input the model to analyze via SBW. Since JDesigner contains no simulation engine, it can pass the model to Jarnac which in turn simulates the model and returns the data. These data will then be displayed via JDesigner. This workflow is displayed in Figure 4.

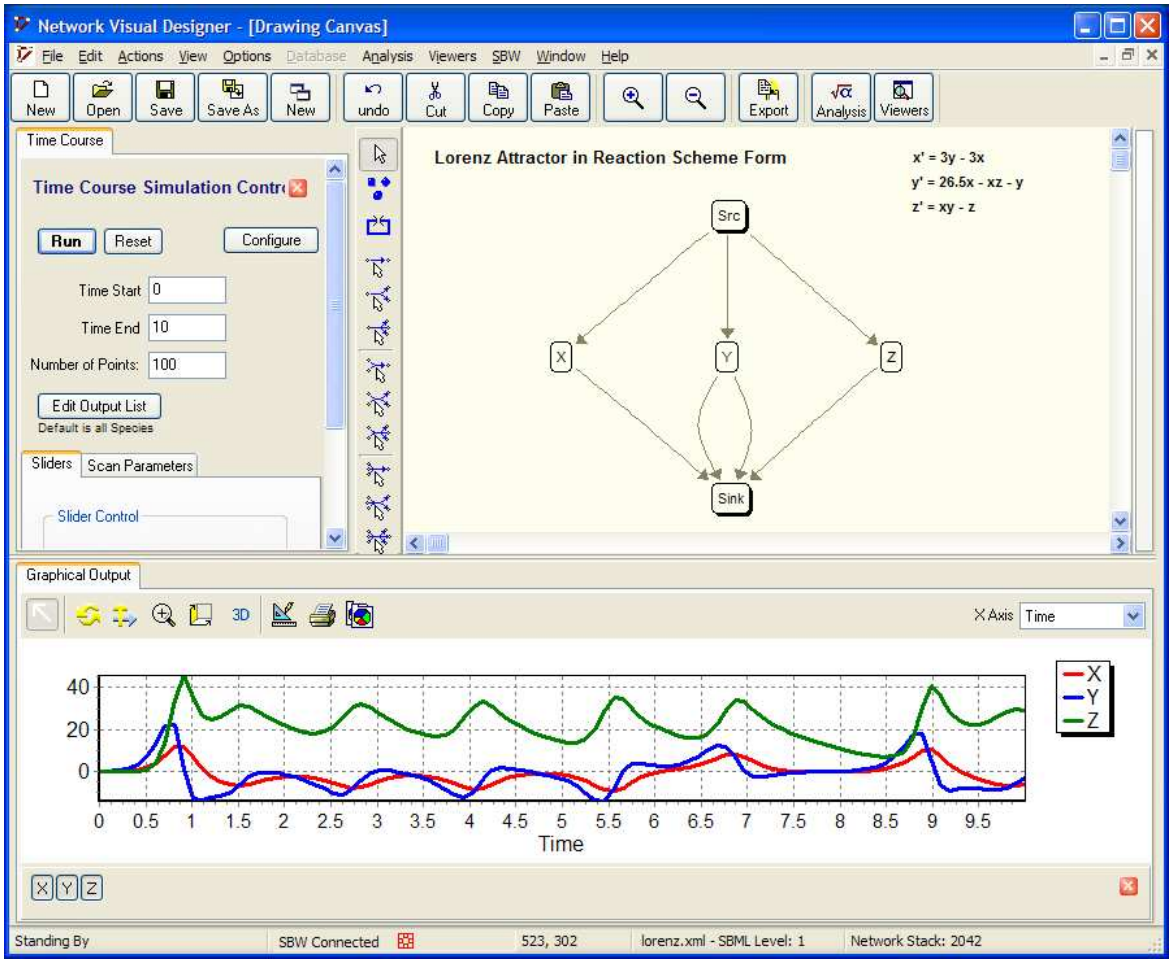


Figure 4: JDesigner using Jarnac to simulate / visualize a Lorenz attractor (Herbert Sauro)

For more information regarding SBML see also chapter 3.2.2.2.

3 Analysis

This chapter will discuss previous work done in the field of reaction-diffusion systems, simulation and visualization. This will help to create a better understanding towards the individual tasks that have to be completed in order configure, simulate and visualize simple three-dimensional reaction-diffusion systems. The next section will detail the structure for this chapter.

3.1 Threefold-ness: Configuration, Simulation and Visualization

This chapter and the following two are structured around the three topics, configuration, simulation and visualization.

The section *Configuration* will analyze common tasks such as inserting an element into a three-dimensional scene, a task very common to modeling applications. In addition, file formats are discussed particular with regard to saving configuration data.

Under the heading of *Simulation,* previous work on reaction-diffusion systems will be discussed, including methods that are available to solve the reaction-diffusion equations, in particular the Euler and Runge-Kutta methods.

The section titled *Visualization* will first take a closer look at available rendering APIs. A second step then takes a look at available volume-visualization libraries. And finally commonly used methods for volume-visualization will be presented.

A final section will wrap up the whole chapter and look at the project as a whole.

3.2 Configuration

The configuration unit will perform mainly three tasks. The first task is to guide the user through the process of specifying the required parameters for the simulation (e.g. dimension of the reaction-volume, the parameters for the reaction scheme to use). A second task is the insertion of species, membranes and channels into the reaction-volume and finally the serialization of the configuration to an external file.

The analysis for the configuration unit will focus on two things. A first part will discuss how common modeling applications handle the task of inserting elements into a three-dimensional space. Thus a comparison of common input metaphors will help to adopt a “natural” input metaphor for this case.

A final part will analyze the serialization of the configuration. It will focus on the comparison of a proprietary file format versus the use of a common exchangeable file format.

3.2.1 Input metaphors

This section takes a closer look how well known modeling applications such as Maya[5] or 3ds max[6] and Rhinoceros[7] treat the tasks of inserting objects into a three-dimensional environment. This will help to provide a solid basis for the configuration of the reaction-volume in this case. The analysis will include the most common modeling operations:

- Insertion of an object – as an example a box was inserted with each of the three programs once into the origin of the scene, and once onto a specific position.
- Changing of parameters – here the previously inserted box was moved around, or the dimensions were changed.
- Deleting of an object – a final task to simply delete the object.

The next three subchapters will explain how each application performed the three tasks. In a comparison-section afterwards will be compiled what common features were and these will be set as goals.

3.2.1.1 3ds max 6

The above specified operations have been tested with a streaming trial version of 3ds max version 6.

Inserting an element with 3ds max 6 works the following way; first the user selects the kind of object to create from an object pane. After selecting the element (here a box) a property pane appears that allows a user to directly specify the values. Alternatively 3ds max 6 uses a three point metaphor, of first selecting two diagonal points on the base surface and then selecting the height.

Modifying the parameters of the box can be done either by changing the property-values in the property pane, or by using move and scale tools.

Deleting works intuitively by pressing delete on a selected object. A script language can also be used to perform all the desired operations.

3.2.1.2 Maya 5 PLE

The operations have been tested on the Maya 5.0 Personal Learning Edition available from the Alias website.

Creating primitive objects with Maya is possible in the following two ways. First of all, one can simply select “Create Cube” which will instantly place a box into the origin of the current scene. There it can be modified either through scale or move tools, or via input in a parameter pane. The other possibility is to select a box item displayed directly behind the “Create Cube” menu. If that one is selected an advanced property panel pops up and allows a user to enter all parameters for the cube.

The box is modified as already mentioned with move or scale tools or through changing parameters in a property pane.

Deleting an element works by selecting an Object (again via mouse input or menu-command) and pressing “Delete” (or selecting the menu command “File\Delete”).

---

[5] Maya is a registered trademark of the Alias Systems Corp.
[6] 3ds max is a registered trademark of the Autodesk Inc.
[7] Rhinoceros is a registered trademark of Robert McNeel & Assoc.

For all options there is also a rich scripting language which makes it possible to write custom scripts that simplify any tasks the user might have.

3.2.1.3 Rhinoceros

All operations were tested on the evaluation version of Rhinoceros version 3.0 released on Nov. 26th 2003.

For inserting boxes Rhinoceros provides three insertion modes. Method one defines the base surface through specifying two diagonal corners and the height through mouse movement. Method two specifies first the center of the base surface then allows defining the dimensions of the base surface through mouse movement, and finally the height is defined again through mouse movement. The last method allows specifying the base surface through three points and finally the height through mouse movement. Alternatively Rhinoceros allows users to specify each point through an input console.

Modifying the parameters of the box for example through scaling or moving follows the same strategy. Either the corresponding action is selected from toolbar or menu, and then specified through mouse input, or console input achieves the same result.

Consequently the delete action is executed by selecting the desired object and pressing “delete” or through selection and deletion of the object using the console.

3.2.1.4 Comparison

After seeing how commonly used applications handle the simple task of inserting and modifying elements this is what seems to be the best way of handling it in this case. First of all there will be specialized insertion tools for each kind of elements that can be inserted (i.e. channels, membranes and species). All tools will work in a similar way. Once activated the insert tool can be moved over the reaction-volume and elements can be inserted on any position (if no other element is present on that location). Alternatively a pane will appear that allows the specification of all parameters through user input.

To modify an inserted object the user will be allowed to select the object by mouse selection. For the selected elements a specialized property pane will appear. This pane will allow moving and deleting the element. Alternatively mouse input will allow changing the properties of an element.

Although all modeling applications supported a script-language there will be no support for a script language for the configuration. A script language for the simple tasks of insertion, modifying and deleting of elements would further complicate the application.

3.2.2 Serialization

This chapter will first state the immediate possibilities for storing the configuration data. Therefore it will look at the following file formats in order to be better able to compare them.

3.2.2.1 Binary File Format

Binary file formats have been extensively used in the past mainly because of two reasons. One reason is the time needed for the execution of the serialization algorithms. It is very fast to block-write huge amounts of data into a file with only a minimum of needed transformation. The other reason is the ease of implementation. Writing or reading files in a binary format is certainly not as demanding for the programmer than parsing / writing xml streams.

The major drawback of binary files is that it is much harder to use the files across different operating systems or even applications. And of course binary data is by definition not human-readable which makes it a bad choice for configuration data. A final drawback is that format changes often mean that a complete rewrite of the saving routine (even if it is a minor change) is necessary.

3.2.2.2 CellML

According to the specification of CellML (c.f. [Cue03]) this language is:

> *… an XML-based language for describing and exchanging models of cellular and subcellular processes. MathML embedded in CellML documents is used to define the underlying mathematics of models. Models consist of a network of re-usable components, each with variables and equations manipulating those variables. Models may import other models to create systems of increasing complexity. Metadata may be embedded in CellML documents using RDF.*

Upon examining the specification closer it turns out that CellML is based upon connected components, whereby components are abstract objects that provide interfaces and hide information. Each component consists of:

- private and / or public interfaces
- unit definitions
- variable definitions (All variables have to be assigned appropriate physical units. This is being done to ensure consistency.)
- underlying mathematics (formulated using MathML[8]) and
- meta-data (usually defined using RDF[9])

To share information across several components, connections are used. These connect the interfaces of the two components and thus allow sharing of information and variables. A model then is for CellML a container for components, connections, units and meta-data. Figure 5 displays the CellML model structure. (The picture was taken from [Nie04]).

[8] MathML is a markup language as defined under http://www.w3.org/TR/2001/REC-MathML2-20010221
[9] short for Resource Description Framework, a language defined in "RDF Model and Syntax Recommendation" for more information see http://www.w3c.org/RDF/ and http://www.w3c.org/1999/REC-rdf-syntax-19990222/

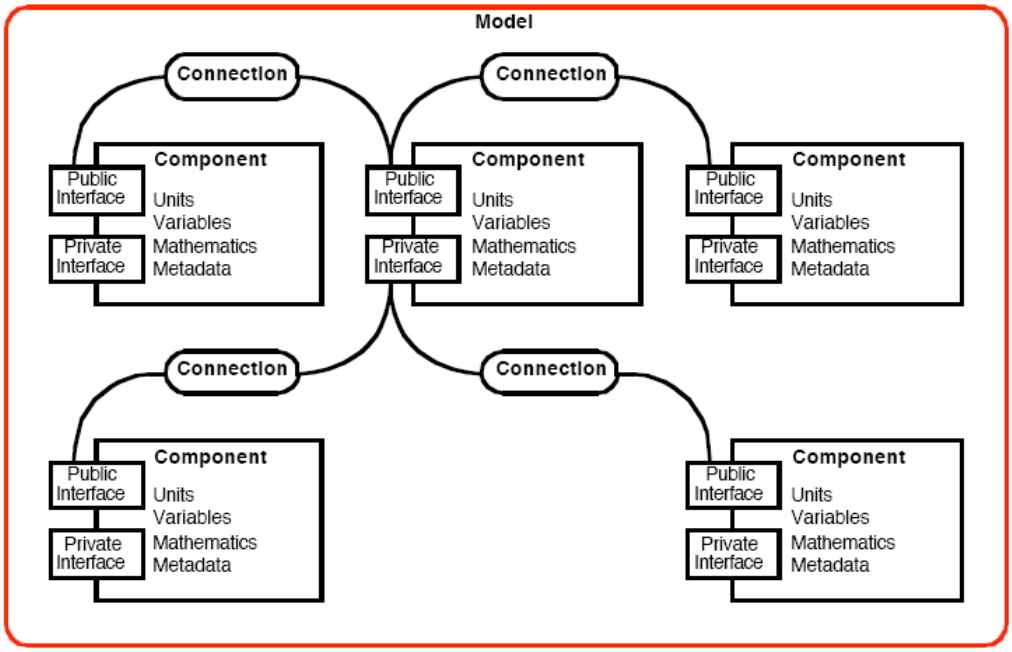


Figure 5: Structure of a CellML model specification (from [Nie04])

According to [Fin04]: "Unlike SBML, CellML explicitly attacks the model integration problem. Like SBML, however, CellML can only encompass a limited range of models that exclude for example, discrete-event systems. CellML is less widely used than the more pragmatically driven SBML".

3.2.2.3 Systems Biology Markup Language (SBML)

The Systems Biology Markup Language (SBML) is a standardized XML dialect for representing models of biochemical reaction networks. This allows for models to be freely exchanged over a wide variety of programs. (An updated list of applications currently supporting SBML can be found at: http://www.sbml.org)

Simple networks of biochemical reactions usually consist of the following components: reactant species, product species, reactions, rate laws, and parameters in the rate laws. To analyze or simulate a network further components have to be specified, such as compartments for species or units for the various quantities. These elements have been captured to build the basic structure of an SBML file (of SBML level 2 c.f. [Fin03a] page 3):

Beginning of model definition:

list of function definitions(optional)
list of unit definitions (optional)
list of compartments (optional)
list of species (optional)
list of parameters (optional)
list of rules (optional)
list of reactions (optional)
list of events (optional)

End of model definition

This structure is meant to support basic biochemical network models and operations available in existing analysis/simulation tools. The SBML standard was developed in the context of interacting

with a number of existing simulation packages. Hence it is widely accepted through a variety of existing tools. For more information about SBML see also [Huc03].

Of course SBML is still work in progress, one of the current shortcomings of SBML Level 2 is, that it is not possible to add spatial information to the model description. SBML however provides a tag called an annotation which permits extensions to be added to SBML. SBML allows each element to be annotated. This enables developers to save structured information until the desired feature (like spatial data) finds its way into the specifications. A good example as to how the annotations can be used to save and load complex layout information can be found in [Sau03b].

In order to simplify the handling of SBML to a vide variety of users a support library was created. This library is called "libSBML" and is available for Java and C++ on Linux-, Windows-, Solaris- and MacOS-systems. This library handles the creation or processing of SBML documents. One very interesting feature is for example the conversion of mathematical formulas from MathML to post-fix notation or vice versa. That simplifies the work with SBML documents a lot. With [Bor03] a comprehensive manual was provided. Recently even an SBW-module was created that provided the libSBML functionality to every other SBW-module. This development may help to further spread SBML throughout the community.

3.2.2.4 Comparison

Given the discussion on binary formats it would seem safe to eliminate this option as a means for saving configuration data. A better possibility for saving the files would be a markup language. This would ensure a human readable[10], interchangeable format. As the analysis of the CellML specification ([Cue03]) showed: the CellML format is quite complex. Furthermore a configuration file represented in that format would not benefit the project much, since hardly any software exists that would provide complementary features. The alternative XML-dialect SBML level 2 seems to be a better choice, since embedding the basic configuration elements into a valid SBML structure (as described in the last section) involves hardly any overhead. A configuration file could be saved using SBML like this:

The configured reaction-volume, in which the reaction-diffusion simulation will be performed, can be seen as a compartment. All additional parameters concerning the whole reaction-volume, such as the diffusion rates of the species X and Y, the reaction that should be performed or the step size of the integrator for the simulation would be stored as annotations of that compartment.

Finally all elements that can be inserted, namely the species X and Y, the membranes or the cannels could be saved as species. For example under the SBML element representing species X there would be annotations for each inserted species X with its position and concentration.

Of course saving the information that way would involve quite some overhead for XML parsing and writing. Also XML documents tend to get bigger than binary documents containing the same amount of information. So what would be the advantages compared to the simple serialization into a binary format. First of all it would be much simpler exchanging the configuration files over different countries or systems. This is due to the fixed UTF-8 encoding of the SBML dialect. Secondly and perhaps more importantly it would allow the configuration to be changed by another application. The idea here would be to create an initial configuration file with the configuration unit and then pass that file on to another application that is able to read SBML. This application might then be able to specify the reactions much more comfortably than it will be possible through the configuration unit.

[10] The author is aware of the fact that very large XML documents tend to get confusing, still in most cases XML dialects are human readable.

3.3 Simulation

Tasks for the simulation unit would be threefold again. First reading and validating the configuration file which will initialize the simulation unit. Then the simulator will continue to perform the next two steps until the user aborts:

- perform a simulation step,
- Return the simulation data / store simulation data to file.

The main focus for the analysis will be the simulation part. The first thing to be researched is previous work on reaction-diffusion systems. Another thing to research would be what kind of integrator a custom written simulator should use. A decision here should be dependent on the factors runtime and stability. Although these factors might be complementary at least the available possibilities should be analyzed.

3.3.1 Reaction-Diffusion Systems

Here previous work on reaction-diffusion systems will be recapitulated. The focus hereby is on researchers that contributed their work on pattern formation in connection to reaction-diffusion systems. Most influential for this work is certainly the research of:

- Alan Turing,
- Hans Meinhardt and
- Greg Turk

This section will also take a closer look at the program “Ilya” - a reaction diffusion simulator written by Herbert Sauro in 1997. As already stated in the problem definition (section 1.2) this program will represent the basis for the simulation unit to be developed here.

3.3.1.1 The chemical basis of Morphogenesis

Most influential for this project is certainly the work of Alan Turing [Tur52]. He was the first to introduce the idea that chemical substances could react with one another and diffuse through an embryo to create stable patters. In his 1952 paper on the chemical basis of morphogenesis he discusses at length the break down and formation of chemicals due to reactions between different molecules in a reaction-diffusion system. He proposes several reaction-diffusion systems and continues to describe one such system with linear reaction terms. This system is then solved in discrete and continuous form. These (discrete) systems are still used whenever it comes to simulate reaction-diffusion. The equations for a two-species reaction-diffusion system for a one-dimensional ring of cells in his paper look like this:

$$\begin{aligned} \frac{\partial x}{\partial t} &= F_x(x, y) + D_x \nabla x \\ \frac{\partial y}{\partial t} &= F_y(x, y) + D_y \nabla y \end{aligned} \tag{1.1}$$

The functions F_x and F_y take the concentrations of X and Y at the given time and let them react according to their respective chemical formulas. The constants D_x and D_y are the diffusity con-

stants of species X or species Y. They determine how fast the species diffuse. The term ∇x or ∇y stands for the neighboring elements of the current *x* or *y*. In the one-dimensional case this will be *x-1* and *x+1* for *x* and *y-1* and *y+1* for *y*. This will result in a change of concentration from a unit with higher concentration to the neighboring unit with a lower concentration.

Finally a dappling pattern produced by the explained reaction-diffusion system is displayed that is reminiscent of the spots on cows. It is presented in Figure 6.

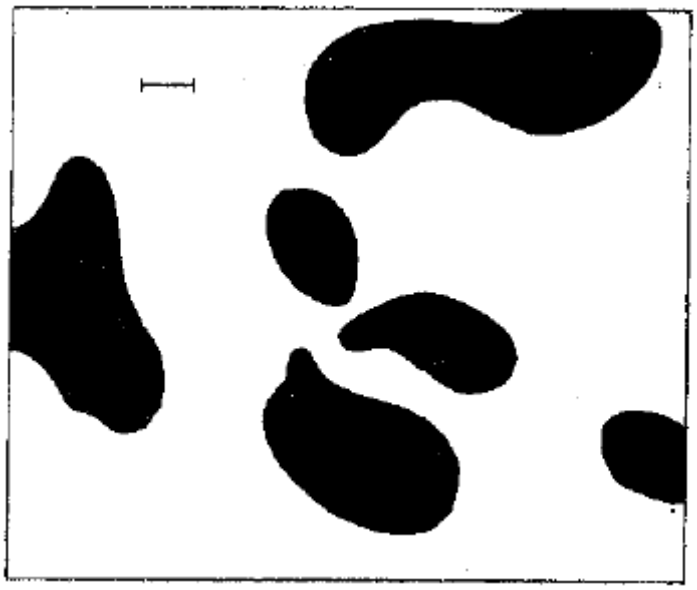


Figure 6: "Dappling" pattern presented in [Tur52]

A continuation of Turing's studies can be found in [Lep03]. Here Leppänen et al. describe the formation of two-dimensional Spatio-temporal patterns employing a generic Turing model. The authors expect this work to bring insight to recent biological finding of temporal patterns on animal skin.

3.3.1.2 Biological principles for pattern formation

In a series of essays Meinhardt describes in [Mei01] the biological principles for pattern formation. In the first part he details the activator-inhibitor reaction. An *activator* in this sense is a substance that "… has a non-linear positive feedback on its own production rate. Its autocatalysis is slowed down by a long ranging *inhibitor*. A necessary condition for the formation of a stable pattern is that the inhibitor diffuses much faster than the activator and has a shorter half life. In other ranges of parameters oscillations and traveling waves can occur. These modes will be discussed further below in connection with the patterns on the shells of mollusks." (c.f. [Mei01] pattern1) He then points out that a pattern emerges whenever the size of the activator-inhibitor-reaction-field becomes larger than the range of the activator. Finally he demonstrates how the nervous system of *Drosophila* can be explained through such a system.

Meinhardt then continues to explain the formation of periodic structures. He mentions for example that stripes, a pattern often encountered, can be formed whenever the "… rate of activator autocatalysis saturates at high concentrations. This leads also to a limitation of the inhibitor production. More cells become activated at a lower level until sufficient inhibitor is produced. In other word, the activated regions have the tendency to enlarge, however, in order to become activated; a close neighborhood to non-activated cells is essential into which the inhibitor can be dumped. Both requirements, large activated patches and a direct neighborhood of non-activated cells, seems to

contradict each other. This is, however, not the case. In a stripe-like activation pattern, each activated cell has an activated neighbor and non-activated cells are close by. Stripe formation is a very frequent phenomenon at very different developmental situations. …" (c.f. [Mei01] pattern 2) In a later part of this chapter Meinhardt then explains how source density might be applied together with the activator-inhibitor reaction to generate structures close to each other. This is explained on the example of head, foot and tentacle formation in the freshwater polyp *Hydra*. For an interesting discussion on the activator-inhibitor theory see also [Har93].

In earlier work Meinhardt described (together with other researchers such as Bard and Murray) how reaction-diffusion systems can be used to create two-dimensional patterns such as spots and stripes. Meinhardt is especially known for proposing a stripe generating reaction-diffusion system. This work was done in 1982 and employs a five-chemical reaction-diffusion system which is stated as this:

$$
\begin{aligned}
\frac{\partial g_1}{\partial t} &= \frac{c s_2 g_1^{\,2}}{r} - \alpha g_1 + Dg\nabla^2 g_1 + \rho_0 \\
\frac{\partial g_2}{\partial t} &= \frac{c s_2 g_2^{\,2}}{r} - \alpha g_2 + Dg\nabla^2 g_2 + \rho_0 \\
\frac{\partial r}{\partial t} &= c s_2 g_1^{\,2} + c s_1 g_2^{\,2} - \beta r \\
\frac{\partial s_1}{\partial t} &= \gamma\left(g_1 - s_1\right) + Ds\nabla^2 s_1 + \rho_1 \\
\frac{\partial s_2}{\partial t} &= \gamma\left(g_2 - s_2\right) + Ds\nabla^2 s_2 + \rho_1
\end{aligned}
\qquad (1.2)
$$

In this set of equations the chemicals g_1 and g_2 represent the present of one stripe color or another (e.g. black or white). "r" is a concentration that makes sure that only one color is present at each individual location, while the chemicals s_1 and s_2 limit the stripes in their width. Meinhardt even gives the source code for simulation of these systems. [Mei82].

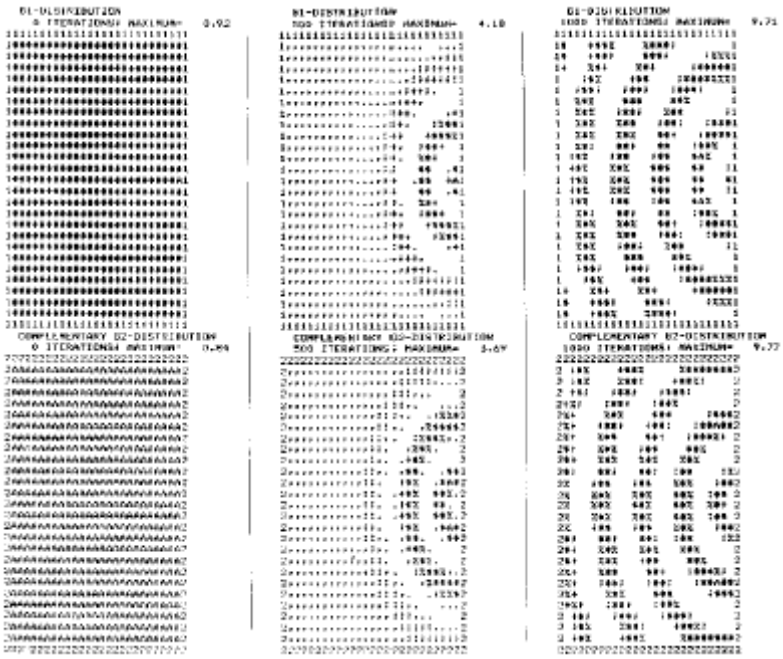


Figure 7: Pattern formation by lateral activation. Stable stripes are formed (from [Mei82])

For a thorough understanding of the activator-inhibitor principle and its connection with Turing's paper the author recommends [Har93].

3.3.1.3 Generating Textures using Reaction-Diffusion

In [Tur91] Greg Turk describes how reaction-diffusion systems can be used to generate textures. He even gives an algorithm to apply these textures on arbitrary surfaces. In fact these reaction-diffusion systems are simulated on the surface making slight changes on the diffusion-coefficients and thus allowing for a smooth texture that is not "stretched" on the surface. This idea is described in more detail in his thesis [Tur92].

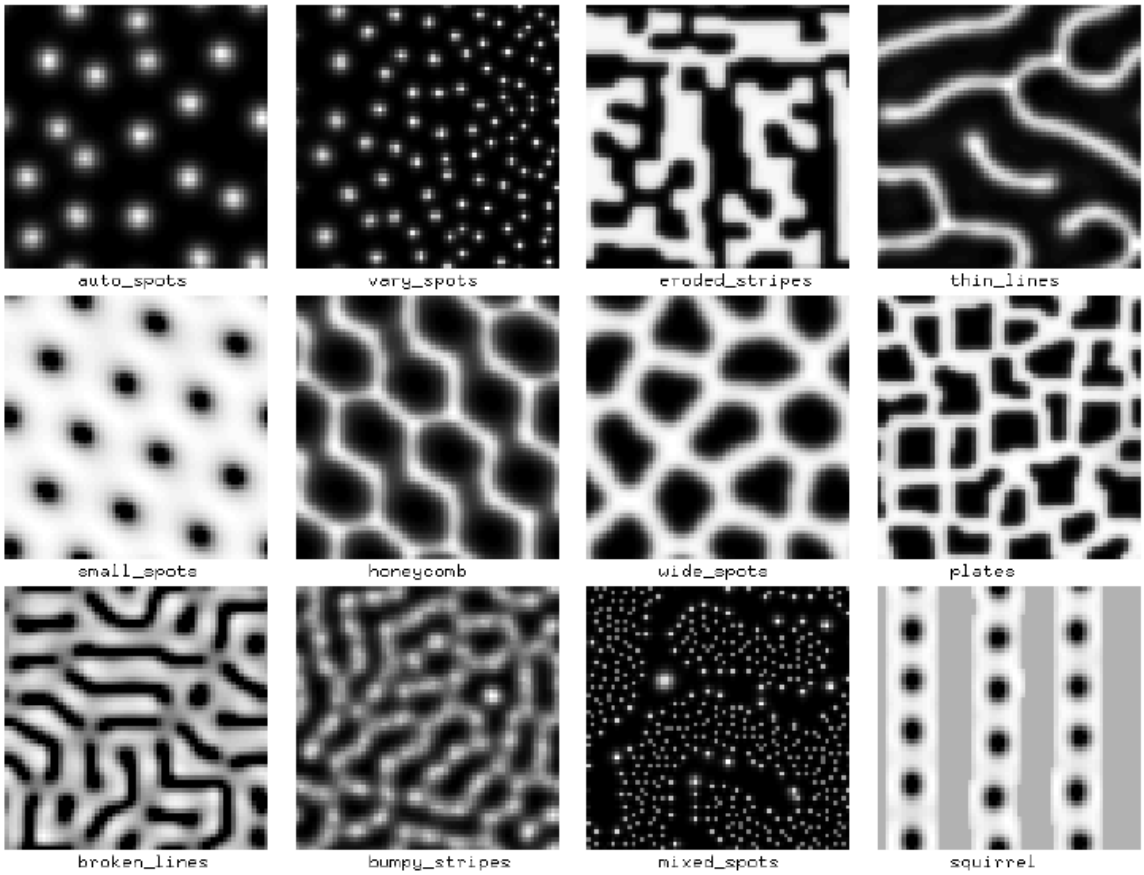


Figure 8: A variety of patterns created with Turk's Cascade system (from [Tur92])

In order to generate all kinds of textures (such as patterns seen regularly on animal skin as giraffes, cheetahs or various fish) a cascade of reaction-diffusion systems is used. This means that after running a first reaction-diffusion system the state is frozen and then these altered concentrations are taken for the initialization of the next reaction-diffusion system. As for the actual reaction-diffusion systems simulated the activator-inhibitor systems as introduced by Turing [Tur52] or Meinhardt [Mei82] are used. Figure 8 displays what patterns are to be expected using cascading reaction-diffusion systems.

3.3.1.4 Ilya – a reaction diffusion simulator

As mentioned before Ilya was written in 1997 by Herbert M. Sauro. It simulates the Brusselator Reaction-Diffusion scheme for two-dimensional reaction-volumes with fixed resolutions: 100x100, 150x150, 200x200 and 300x300 elements.

Ilya employs a simple Euler integrator that solves the following equation for each cell element:

$$\begin{aligned} \nabla^2 x &= x_{i+1,j} - 4x_{i,j} + x_{i-1,j} + x_{i,j+1} + x_{i,j-1} \\ \nabla^2 y &= y_{i+1,j} - 4y_{i,j} + y_{i-1,j} + y_{i,j+1} + y_{i,j-1} \\ x_{change_{i,j}} &= \left(A + x_{i,j}{}^2 y - Bx_{i,j} - x_{i,j}\right) + rD_x \nabla^2 x \\ y_{change_{i,j}} &= \left(Bx_{i,j} - x_{i,j}{}^2 y\right) + rD_y \nabla^2 y \end{aligned} \tag{1.3}$$

The terms in the brackets represent the reaction equations that can optionally be disabled in order to simulate diffusion only. For more information on the Brusselator scheme (c.f. [Nic77]) see also section 4.3.1.2. The constant *r* represents a scaling factor that determines how strongly diffusion influences the whole update step. These changes will be applied to the original data set during the integration step. And another simulation run will be performed.

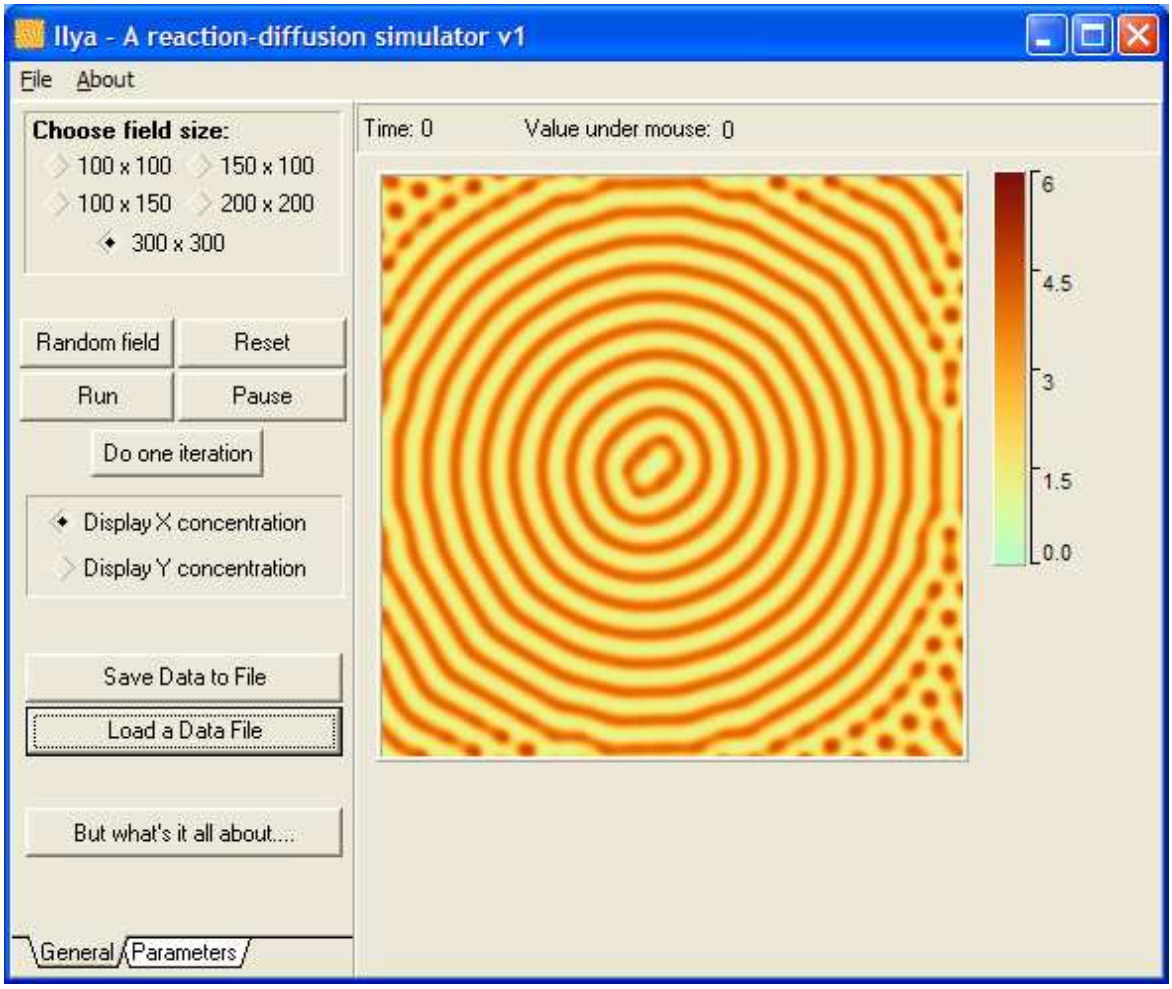


Figure 9: Screenshot of ILYA showing a simulation run (300x300)[11]

[11] ILYA copyright 1997 Future Skill Software

Ilya allows the configuration of the initial concentration set or the changing of the current concentration set at each point in time. Therefore a new concentration is simply “painted” over the old one. This concept can’t be applied to the three-dimensional case though.

3.3.2 Integrators

The most time consuming part of a reaction-diffusion simulator is the integration of the reaction-diffusion equations. Two main issues arise in the choice of a suitable integrator, the stability of the method and the time it takes to integrate. A key parameter in any integrator is the step-size. This is the time step taken to move to the next time point. The higher the step-size the faster the integration will advance, however this also leads to greater instability in the solution. In this project three integrators were considered:

- Euler integrator
- 2nd order Runge Kutta Integrator
- 4th order Runge Kutta Integrator

The Euler integrator uses the formula:

$$y_{n+1} = y_n + stepsize \cdot f\left(x_n, y_n\right) \tag{1.4}$$

Using Euler integration the derivative information is only evaluated once at the beginning of the interval. Thus the step’s error is of order

$$O\left(stepsize^2\right) \tag{1.5}$$

. Although the Euler method is fairly simple to implement it is less accurate and less stable than the other methods mentioned here.

The second order Runge-Kutta method (or midpoint method) is defined through the formulas:

$$\begin{aligned} k_1 &= stepsize \cdot f\left(x_n, y_n\right) \\ k_2 &= setpsize \cdot f\left(x_n + \frac{1}{2} stepsize, y_n + \frac{1}{2} k_1\right) \\ y_{n+1} &= y_n + k_2 + O\left(stepsize^3\right) \end{aligned} \tag{1.6}$$

This method has second-order accuracy (obtained through using a first derivative to find the midpoint, and then the derivative of the midpoint to get the next value).

The most often used formula is the fourth-order Runge-Kutta formula which uses the following set of formulas:

$$
\begin{aligned}
k_1 &= stepsize \cdot f\left(x_n, y_n\right) \\
k_2 &= stepsize \cdot f\left(x_n + \frac{stepsize}{2}, y_n + \frac{k_1}{2}\right) \\
k_3 &= stepsize \cdot f\left(x_n + \frac{stepsize}{2}, y_n + \frac{k_2}{2}\right) \\
k_4 &= stepsize \cdot f\left(x_n + stepsize, y_n + k_3\right) \\
y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O\left(stepsize^5\right)
\end{aligned}
\tag{1.7}
$$

This method would be superior to the midpoint method, if a twice times larger step-size can be used - which is often the case.

In order to improve Runge Kutta integrators further a next step would be to apply adaptive step-size control. Here the basic idea would be to measure the error of the integration by taking a step twice, once as a full step and next independently as two half steps. The difference of these two evaluations can then be applied (together with a confidence parameter) to calculate a new step-size.

Other integrator methods like Richardson-extrapolation and predictor-corrector methods have been rejected because (i) in this case evaluation of the reaction-diffusion function is rather cheap and moderate accuracy (smaller then 10^{-5}) is sufficient (because the computed values will be needed only to perform a color lookup) and (ii) the function will be evaluated by table-lookups.

For further discussion of these integrators as well as Richardson extrapolation and predictor-corrector methods see [Pre99]. An interesting description on how to achieve a stable simulator can also be found with [Mon02].

3.4 Visualization

The fourth part of this chapter will deal with the analysis for the visualization of the simulation data. Basic tasks for this unit will be acquiring the simulation data, preparing the data for display, and finally rendering of the data.

What has to be considered here is what rendering API to use, what sorts of visualization algorithms should be used in the visualization process, and what sort of visualization libraries are already available and would suit this project.

3.4.1 3D Rendering APIs

This section will take a closer look at the most common rendering APIs - namely OpenGL and Direct3D. Furthermore an OpenGL library called OpenSceneGraph will also be analyzed as it gives OpenGL advanced features that might be vital for this project.

Another rendering and scene-graph API will not be discussed here – namely Java3D. Although that API is known for its thorough specification and platform independence it turned out in a pre-

vious project[12] that a huge amount of memory is needed and an overall slower runtime (compared to a project implemented using C++/OpenGL) is to be expected. Since it is to be expected that the simulation data will already place a big stress on the memory (the reason for that lies in the three-dimensional structure of the reaction-volume) Java3D seems not to be the right choice.

3.4.1.1 Direct3D

DirectX is a set of low-level APIs that contains components for two- and three-dimensional graphics, sound effects and music, input devices and networked applications. One advantage of using DirectX is that all of the low-level APIs follow the same concept. Normally a developer would encounter many different APIs following different software design patterns. Direct3D was acquired by Microsoft in 1996. The API is defined through its COM interfaces, help files and sample code.

DirectX is an integral part of the Windows core. Since all modern graphics cards come along with DirectX drivers very good performance can be expected. This strong support from graphics card manufacturers leads to the fact that new features will be added to DirectX regularly. Recent additions to the graphics API were a shading language and changes in animated meshes.

The Direct3D core follows the traditional rendering pipeline but allows programmable extensions for pixel shaders (controlling pixel shading) or vertex shaders (manipulating object geometry). Finally the current version of Direct3D (version 9.0c) allows also for the following features:

- bump mapping (allows to simulate a rough surface in 3D scenes),
- anti-aliasing (allows to reduce the appearance of stair-step pixels),
- vertex blending (allows to create effects such as smooth joints and bulging muscles in character animations) and
- tweening (allows blending of two provided position or normal streams)

An additional interesting feature is multiple monitor support. For details see also [Mic02].

3.4.1.2 OpenGL

OpenGL (standing for “Open Graphics Library”) was developed as an industry standard by the OpenGL Architectural Review Board (ARB) after the initial version was developed by SGI. Founded in 1992 the ARB controls the definition of OpenGL. The current version 1.5 was released on July 29th 2003. One big advantage for using OpenGL is that is was built for compatibility across hardware and operating systems. Thus an OpenGL application might theoretically be ported to any system.

OpenGL is basically a software interface for graphics hardware that renders multidimensional objects into a frame-buffer. It can be seen as a state machine whose attributes control how sets of vertices and normals are to be rendered. The programming model used for OpenGL is through include- and library-files.

Interesting features for OpenGL are:

- support for picking
- hardware independent z-buffer access and

[12] This hints at the project “Pacman3D” (c.f. [Pac02]). Here the game “Pacman” was reimplemented based on a 3D environment using Java3D. There memory demands and runtime of Java3D was much slower than what could have been achieved using C++ and OpenGL.

- accumulation buffers

Although OpenGL does not support as many functions as DirectX there exist a multitude on utility libraries that extend the features of OpenGL.

On September 7th, 2004 the ARB released the specification of OpenGL version 2.0. The main changes involved are:

- programmable shading is now supported with the OpenGL Shading Language
- multiple render targets are supported
- textures no longer are restricted to textures with power-of-two dimensions
- Separate stencil functionality might be used for front and back faces of primitives. With that step performance of shadow volume and Constructive Solid Geometry rendering algorithms is improved.

For the full specification see [Seg04].

3.4.1.2.1 OpenSceneGraph

One high-level API built on top of OpenGL is OpenSceneGraph (OSG). It is an object oriented framework built on top of OpenGL. Its intention is to "free the developer from implementing and optimizing low level graphics calls, and provide many additional utilities for rapid development of graphics applications." (c.f. [Bur04]) OpenSceneGraph was started by Don Burns in 1998. The aim of the project at that time was to port a Performer based application from IRIX to a Linux PC. Robert Osfield began 1999 to port the scene-graph element to Windows. Since then a lot of work has been put into a project and it has gotten a lot of attention throughout the world. The current version is 0.9.7.

OSG as the name implies implements a scene-graph API. A scene-graph is an acyclic, directed graph of nodes. Nodes in this regard correspond to objects like triangle meshes, lights or transformation objects. An image is rendered by traversing the tree. Thereby each node affects the rendering process. Thus the traversing of the tree can be seen as changing the states of the OpenGL state machine.

The major drawback of OpenSceneGraph at the present state is documentation. Available documentation is mostly restricted to source code, simple examples and support through a news-group. Nonetheless the features of OSG are convincing:

- an extensive scene-graph API
- support for various file formats
- support for picking
- platform independency

3.4.1.3 Comparison

After introducing these available three APIs this section evaluates how these APIs perform. Therefore a small visualization application was implemented for each of these APIs to be able to compare them. The sample application worked on a three-dimensional field of concentrations (double values) of size 50x50x50. For each element a cube of unit length was generated. For each cube a color representing the concentration was assigned. Furthermore the overall position of each cube was determined by the position of the concentration element in the three-dimensional data-set.

Here the results for the categories: **implementation time**, **memory consumption, cooperation with the window toolkit** and **overall runtime**:

Implementation time: The implementation time is a critical factor for a project like this since the implementation period of the thesis follows a strict schedule. As expected writing the application for the high-level scene-graph API OpenSceneGraph was done very quickly. Ignoring all higher level functions and using only low level OpenGL functions took somewhat longer. The end of that scale was represented by Direct3D. The reason for that was using the COM concept. Although Direct3D is thoroughly integrated into the Windows core, even supported with Wizards by Visual Studio (the Wizard already sets up the display context inserted an example mesh) writing the actual visualization core took much longer compared to OpenGL and OSG.

Memory consumption: As already indicated at the beginning of the chapter, memory consumption may turn out to be the deciding factor. With cubic reaction-volumes storing at least the concentrations a large amount of memory is already allocated and can't be released. In this category the results showed the opposite result than the last heading. Under Direct3D a constant amount of memory was needed. A slightly higher amount of memory was needed for OpenGL although here too the amount needed was constant. OpenSceneGraph was at the end of the scale. Although OSG provides for reference counting and thus rudimentary memory management there seems to be an issue with releasing that memory. The memory was released though not at expected times. This delay leads from time to time to instabilities of the application.

Cooperation with the windowing toolkit: Of course the rendering window won't stand alone it needs to be accompanied by a user friendly GUI. Thus under this heading it will be compared how the individual APIs work together with windowing toolkits (windowing toolkits tested here were Gtk+, Qt, FOX, MFC and Win Forms). The platform independent API OpenGL works seamlessly with every window toolkit. Each toolkit had a dedicated object that provided the GL context. OpenSceneGraph that is based on OpenGL could also be made to work with any of these toolkits. The integration is not as seamless as with OpenGL since the existing toolkit objects have to be altered slightly. Helpful here are the support on the OSG mailing list, or example implementations on the OSG web site. Direct3D as a proprietary API was supported only by MFC and Win Forms. The support here was very convincing though; Wizards helped setting up the rendering context and even generated example source code.

Runtime: The importance of the overall runtime needed to generate a frame can readily be seen. Of course this factor will be dependent on other factors such as the memory consumption. Astonishingly here all three APIs lay very close together. There was a slight better performance for Direct3D, followed by OpenGL and lastly OpenSceneGraph.

Overall OpenSceneGraph will be chosen for this project. The main reason for this is the ease implementing the concepts, while maintaining the possibility of platform independence. Additionally OSG supports features like picking or support for many file-formats out of the box. This will help to save much time. Furthermore as OSG is based on OpenGL it would always be possible to fall back to OpenGL just in case, that a certain visualization type would not be possible with OSG due to memory or runtime restrictions. Finally in a prior project (c.f. [AMI04]) good experiences were made using this API.

3.4.2 Volume-Visualization Algorithms

In this section the most common volume-visualization algorithms will be considered. As mentioned in section 2.2 classic volume-visualization algorithms can be divided into the three categories:

- direct volume rendering,
- interactive methods
- surface fitting algorithms

This categorization shall be used to further structure this section.

3.4.2.1 Direct Volume Rendering (DVR)

Algorithms falling in this category directly render the data elements into the screen space, without creating geometric objects for them. So for each rendered image (i.e. each time the view has been changed) the whole dataset will be traversed. DVR algorithms achieve good results visualizing gases, fluids or clouds. Common to DVR algorithms is that the user will first have to specify a color map and opacity map for the data. The most important algorithms in this category are:

- Ray-casting

  Ray casting is the most used DVR algorithm for high-quality images. The algorithm will cast rays for each pixel. If a data-point is hit, it gets the assigned color information. Should color information be present the values are accumulated. A ray stops if it leaves the data-area or the accumulated color information reach a maximum value. There are various versions of this basic algorithm. These are mainly concerned with how the color and opacity information are gathered once a ray hits an object.

  In [Frü96] is described how a pseudo-color approach can be used to apply ray-casting to the visualization of vector fields. The usual representation for vector fields was the introduction of objects tangential to the vector field.

  Kaneda et al. describe in [Kan96] an approach to the problem of improving interactive volume rendering with adjustable color maps. Here basis images are calculated based on a fixed viewpoint and independent from a color-map. Using these basis images allows for interactive change of color maps. These will applied only to the basis image and thus allow for a interactive observation system.

- Splatting

  Splatting will first compute each voxel's contribution to the image and accumulate them. Each voxel will then be given a color an opacity according to the user defined maps. Then they are projected into the image space (using Footprint methods). The contribution of a voxel will be higher the closer it is to the center of the data projection. See also [Enc97] and [Owe99] for details.

- V-buffer rendering

  This algorithm is an extension of the Z-Buffer algorithm to volume-visualization. It was described in detail in [Ups88]. Here the cells of the dataset will be processed into which a cell projects before moving into the next cell. The collected data will be accumulated until either opacity reaches one or the entire cell has been traversed.

3.4.2.2 Interactive Methods

Traditionally algorithms in this category were used to provide the user with a possibility for the efficient exploration of large datasets. Thereby a loss of information was accepted as long as an interactive speed could be achieved. The most common algorithms are:

- Wireframe Contours:

  As implied by the name this algorithm generated lines for the desired iso-value only. While this algorithm is undoubtedly one of the fastest algorithms it displayed minimal information in the generated image.

- Sweeping plane

  For this algorithm the volume was rendered transparent. The user could specify one or more pseudo-color sweeping planes through this volume. This made it a very good algorithm for the interactive exploring of datasets. Though of course the information the image provided was limited to the colors on the “sweeping”-planes.

- Maximum Voxel

  The idea for this algorithm is to resample the maximum value behind each pixel. Therefore nearest-neighbor resampling was used. While this algorithm still achieved interactive speeds it allowed for no shading and grayscale display only.

3.4.2.3 Surface-fitting algorithms (SF)

Surface-fitting algorithms allow the user to input a threshold value. This value will then be used to find a surface in the dataset which will then be displayed. As opposed to DVR methods a geometric model representing the object will be created. Thus each new view can be created very fast without another traversal of the original dataset. A new traversal will only be needed if a different threshold is chosen or a new data set has to be displayed. One assumption is made by the algorithms presented here; it assumes that the data are presented in a structured grid. This assumption should not be a difficulty however, since each collection of data can be represented as a structured grid. Instead of using these Cartesian lattices Carr et al. argue in [Car03] that it would be more efficient to use body-centered cubic lattices. Although here too a large number of cells is involved it is shown that their model is competitive to the below described Marching Cubes algorithm on Cartesian grids.

The most prominent surface-fitting algorithms are:

- contour tracking,

  The basic idea behind contour tracking is the following. First two slices of data are read into the memory. As a next step the algorithm detects a closed contour in respect to the threshold value inputted by the user for each slide. Then the contours in adjacent slices are connected (usually by triangles) and rendered.

- cuberille / opaque cubes,

  The “Cuberille” algorithm also known as opaque cube algorithm was first introduced by Herman and Liu in [Her79]. In a first step the algorithm classifies all data points as either belonging to the object or not. In a next step all voxels (generated from the data points) belonging to the object are rendered. Thus the surface is defined by the surfaces of the outer voxel. Since the number of generated surfaces is immense (because of the fact that cubes are generated for each data point belonging to the object) this algorithm is hardly used.

- marching cubes,

  The Marching Cube (MC) algorithm is one of the most common used volume-visualization algorithms. MC works on the assumption that the data are presented in a structured grid (for example a three-dimensional array). The first four slices of the data-set a read into the memory. In a next step grid cells are examined that contain four vertices from both middle slices (slices 2 and 3 of the four slices taken in the first step). According to the value of the data set and the selected threshold each vertex will now be classified as inside or outside of the object. Based on this classification one can then decide how many triangles are needed for the iso-surface. This is being done by using a table lookup on a table containing the edge information. The exact positions of the triangle-vertices on the edges are next calculated via linear interpolation. From these positions the normals on these vertices cab are determined. These steps will be continued until the whole volume has been processed. First described was this algorithm in [Lor87]. Good explanations can also be found in [Enc97]. See [Bor97] for a complete representation of the edge tables along with some example applications.

- marching tetrahedra

  The marching cube algorithm that was previously described has a problem with ambiguity in special cases. If one cube has marked vertices diagonal to unmarked vertices it cannot be decided whether to separate these vertices or to connect them. Marching Tetrahedra is one possibility to clarify these ambiguities. Each cube is divided into tetrahedra. It is no problem to separate marked and unmarked edges by a single plane in tetrahedra. Similar to the marching cube algorithm, marching tetrahedra also works via classification of the possible triangulations (again via table lookup). The only drawback of this algorithm is that through introduction of edges inside a cube twice as many triangles will be generated than with the marching cube algorithm.

  To further improve the algorithm [Zho97] describes a multi-resolution approach. Here the volume data is subdivided recursively and represented by binary trees. Using an error based model these trees are used by recursively fusing different levels together. This reduces the number of voxels needed to construct the model while promising to preserve the topology of the model.

- dividing cubes,

  Since the marching cube algorithm often generates triangles that are smaller than a pixel there is another algorithm exploiting that fact – the dividing cubes algorithm. Dividing Cubes is similar to Marching Cube in that the user still specifies a threshold value for which a surface is searched. Dividing cube begins with the classification of the vertices of each cube. In the case that this classification indicates a cut with the surface the algorithm projects the cube into the image space and checks for the size. Should it be larger than a pixel, the cube will be divided into eight sub-cubes. Otherwise it will directly be drawn (projected into image space) via for example the painters-algorithm or z-buffer algorithm. Thus dividing cubes does not compute real surfaces. Rather directly “surface points” are created. This also leads to the act that the surface has to be recomputed each time a different view is selected. For a more detailed description of the dividing cubes algorithm see [Ebe04].

3.4.3 Volume-Visualization Libraries

This section will analyze what libraries and toolkits are currently available for volume-visualization. This analysis will take in regard commercial as well as open source products.

- AVS/Express

  According to the vendor, AVS/Express is the world's leading visualization toolkit. In the white paper [AVS04] some of the features of AVS/Express are detailed. Among it is stated that over 800 graphical operations are supported. It was developed to meet the design goals of supporting parallel rendering on PC Windows architecture, supporting all display configurations (such as CAVE, HoloBench, PowerWall and such), supporting any application domains (CFD, Telecoms, Oil & Gas, Medical among others), allow using a component library for deployment and creating a visualization environment structure instead of just a low level graphics library.

  Interesting about AVS/Express is that it supports high-level primitives. These primitives manage raw triangle data that would be used over and over. Otherwise a visualization of large isosurfaces would result in one OpenGL command after each other and then disregarding these information. These high level primitives exist for tri-stripes, tubes and bitmaps.

  The only demonstration application that was available to the author proved that examples throughout various domains could be rendered convincingly even on average desktop-computers. Sadly it gave no real insight about the internal structures of AVS/Express. The author has no doubt however that this toolkit would prove a valuable asset in displaying three-dimensional reaction-diffusion systems.

- IRIS Explorer[13]

  Another commercial visualization library is the IRIS Explorer. The vendor describes the IRIS Explorer as follows: "IRIS Explorer is a development tool that enables you to easily build visualization applications." (from IRIS Explorer Demonstration).

  Instead of a programming library the IRIS Explorer is a visual tool that works in the following way. Datasets, visualization methods and rendering devices are represented as modules (new modules can be written in FORTRAN, C and C++ to enhance existing capabilities). These modules will be selected from a librarian and placed on a map editor. There the individual modules will be visually connected. This will result in the visualization application.

- VisiQuest

  VisiQuest[14] represents the last volume-visualization tool that has been analyzed from the category of commercial volume-visualization libraries. Similar to the IRIS Explorer it also provides a dataflow based application interface that allows for an easy way to develop visualization applications. Here about 300 different visualization modules are available. VisiQuest is available for Unix platforms, Windows systems and Mac OSX.

- OpenDX[15]

  OpenDX represents the open source version of the IBM Visualization Data Explorer. It provides the developer with two possibilities either it can be chosen to use a workflow tool similar to the one described for the IRIS Explorer. Or the developer can access the available visualization modes through accessing the library directly. This makes it an interesting choice for all kinds of volume-visualization. The only drawback here is that a Windows version is only available through Cygwin. And even then a X-Server is necessary. This makes OpenDX not really usable in a Windows environment.

[13] Information about the Iris Explorer was obtained by visiting: http://www.nag.co.uk
[14] VisiQuest is a trademark of AccuSoft Corporation
[15] Information about OpenDX can be obtained from: http://www.opendx.org

- vtk[16]: short for Visualization Toolkit is a freely available C++ class library for 3D graphics and visualization. The idea behind vtk is to write a fully object oriented toolkit, that abstracts from the window toolkit on one side and the rendering API on the other. Instead of the scene graph paradigm, vtk follows the data-flow paradigm. The vtk supports visualization techniques for scalar, vector and tensor visualization. Furthermore modeling algorithms such as decimation, implicit modeling, extrusion and texture cutting. Finally for creating visual displays it supports Delaunay triangulation, splatting and glyphing. For further information about vtk see [Kit04] and [Sch96].

  One example for the usage of vtk in research can be found in [Pek01]. In [Ahr00] the problem of the scientific visualization of extremely large time-varying datasets is being addressed. Here too vtk is being used to provide a scalable, portable way for achieving these visualizations.

- OpenVL[17]: OpenVL is a utility library for handling volume data. In its current stage it supports handling, mapping and manipulating volume data. Volume-visualization support is planned in a later stage.

3.5 Conclusion

This final section will conclude the chapter "Analysis". The last three sections described previous work needed to configure, simulate and visualize three-dimensional reaction-diffusion systems. The task for this section is now to show the connections between the individual parts.

The analysis of different modeling application has shown a way of inserting elements into a three-dimensional space. For this project this three-dimensional space is the reaction-volume that is to be populated by concentrations of two kinds of elements, membranes and channels. This configuration of elements will in a next step be serialized into an external file. Therefore the serialization into a binary format, the CellML format or the SBML format was considered. Because of the straight-forward specification and strong support by other applications SBML seemed to be the right choice as the storage format for the configuration data.

This configuration data is passed on to the simulation-core. The third section of this chapter discussed what sort of system the simulation-core should simulate. Here the system proposed by Alan Turing in [Tur52] seemed to be the right choice. Although Euler Integration is known to make errors on larger step sizes it might work fine in this case. Here parameters will be restricted to a workable range. It seems to be more important for this case to ensure fast integration that works for most cases. Should the integration fail, a mechanism has to be provided to lower the integration step size. After that a successful simulation should be achievable in every case and end with the generation of simulation-data.

What is left is the display of the simulation-data. Therefore the fourth section of this chapter discussed the available rendering APIs. OpenSceneGraph as high level language seemed to be the right choice and was selected. What was left was to examine common volume-visualization algorithms. Here several algorithms from all three categories (direct volume rendering, interactive methods and surface-fitting algorithms) should be implemented. A final part of the fourth section presented some available volume-visualization libraries. This was done to check common features in these libraries and to think about a possible integration of these libraries into this project. The

[16] Details can be found under: http://public.kitware.com/VTK/
[17] See also http://openvl.sourceforge.net/

only visualization library usable in a Windows environment that has a chance of being integrated seems to be vtk.

4 Conception

After the last chapter analyzed what has been done before, this chapter will create a concept for the project. This chapter will define what has to be done in order to configure, simulate and visualize simple three-dimensional reaction-diffusion systems. This conception will be based on the problem stated in section 1.2. The structure of this chapter will be defined in the next introductory section.

4.1 Threefold-ness: Configuration, Simulation and Visualization

As the last chapter, this chapter will be separated into the sections configuration, simulation and visualization. The interconnectedness of the individual parts will not be forgotten. Therefore the problem stated in section 1.2 will be examined closer and requirements compiled, that each of the parts has to fulfill, in order to find a solution for the problem. For the individual parts that means the following.

4.1.1 Configuration

For the configuration part the problem states that:

> "… a *prototypic 3D configuration tool for the initial state of the system has to be developed. This basic tool should enable the user to define and store the location of molecules, membranes and channels within the reaction space of user-defined size. A suitable data structure has to be defined for the representation of the reaction space. …"*

This statement can be boiled down to the following three requirements:

- First of all the user has to be able to specify and configure the reaction space. Important features here would be for example the dimensions of the reaction-space. Furthermore, in case the reaction-diffusion equation to simulate uses any kind of substrate, this also should be configurable.
- Second the configuration unit has to allow for the insertion of channels, membranes and species at various locations. Furthermore additional properties of the elements might need to be set, such as the concentration for species, or the dimension of a membrane.
- Finally this structure has to be stored in a suitable data structure, or perhaps serialized into a file for later reuse.

Section 4.2 will detail this further.

4.1.2 Simulation

For the simulation part the problem is formulated as:

> *"…a 3D simulation algorithm (PDE solver), based on an existing 2D-simulation algorithm for reaction-diffusion systems written by Prof. Herbert Sauro, has to be developed. …"*

Since "simple three-dimensional reaction-diffusion systems" should be simulated, this means that the following tasks should be met by the simulation unit.

- The simulation unit should provide different simulation modes, so that different reaction-diffusion systems can be simulated. "Simple" is hereby interpreted as reaction-diffusion systems involving two species interaction only. Parameters for the simulation mode to use should be specified by the configuration unit. This way the actual simulation-core does not require user feedback.
- Also meta-information about the simulation should be provided by the simulation unit. This way a previous simulation can be continued at a later point in time.

These points will be elaborated in section 4.3.

4.1.3 Visualization

For the visualization part the following problem has been defined:

> *"The main focus of this thesis is the specification and prototypic implementation of a suitable reaction space visualization component for the display of the simulation results. In particular, the possibility of 3D visualization during course of the simulation has to be investigated."*

This leads to the following two requirements:

- In order to find a "suitable reaction space visualization" different visualization modes should be available. Thereby a selection of the most common volume-visualization algorithms (as described previously under section 3.4.2) has to be available.
- To investigate the possibility of three-dimensional visualization during the actual simulation runs the visualization unit should be able to work in two different modes. A first mode will just display previous simulation results and a second mode will alternate with the simulation unit to visualize the results during the course of the simulation.

This will be described further under section 4.4.

4.2 Configuration

As has been stated in the introduction to this chapter (section 4.1.1) the tasks for the configuration unit will be threefold. The first task is to guide the user through the process of specifying needed parameters for the simulation (parameters for the reaction-volume, the parameters for the reaction-scheme to use). A second task is then the insertion of species, membranes and channels into the reaction-volume. And the third part is the serialization of the configuration to an external file.

This subchapter will begin to explain all required elements of the configuration. After that a concept of the configuration unit should follow.

4.2.1 Elements of the configuration

While all other elements are optional there is one element that each configuration needs to have in order for simulation and visualization to work:

- **Reaction-Volume**: the reaction-volume specifies first of all the space in which all reactions will be performed later on. The most important parameter of the reaction-volume is certainly the dimension of it. This is the parameter that decides the overall speed and memory consumption. The dimension will be specified by its width, breadth and depth. Of course two-dimensional simulation and visualization are also important features for the user. This is possible through setting the depth to 1.

  Along with the reaction-volume the user will be asked to specify the simulation mode. This also influences the reaction-volume in the case, that the simulation mode requires the reaction-volume to contain a certain substrate. Since only "simple" reaction-diffusion systems will be considered two assumptions will be made. The first one is that all substrate elements are contained throughout the whole reaction-volume in the same concentration. The next assumption is that although a closed reaction-volume with fixed width, breadth and depth is defined the actual reaction-volume will be a hyper-torus. This means that all borders of the reaction-volume will be connected. In other words, should one element be on the verge of "diffusing out" of the reaction-volume, it will be inserted again at the opposite border. The reason for this is the flux preservation law stating that the outgoing flux is as high as the incoming flux.

All other kinds of elements will be optional. They may or may not be inserted into the reaction-volume. Should the user decide not to insert any of these optional elements the configuration is still valid. Not specifying these elements will result in the assumption that the concentration of the species X and Y is equal to zero throughout the whole reaction-volume, furthermore no membranes and channels will be present. Optional elements that may or may not be added to the configuration are:

- **Molecules**: (in this thesis also called chemicals or species) can be added to set the initial concentration of X or Y at certain positions. On all other positions[18] of the reaction-volume the initial concentration of the respective species is assumed to be equal to zero. The assigned concentration for the species will lie in the range [0…6.0]. Although it will be possible to insert values outside this scale this might influence the performance of the simulation. The reason for that lies in the integration unit as was described in section 3.3.2.
- **Membranes**: are defined by a position of the center, and the dimension. Membranes will hinder (reaction-) diffusion of chemicals through them. Since the scope of this thesis is to simulate "simple" reaction-diffusion systems the presence there will be no permeable membranes. The author is aware of the fact that this restriction is a strong one. In biological systems it

[18] Regarding the problem of concentrations at certain positions the author wholeheartedly agrees with [Tur52] who pointed out that: "…[the] description of the system in terms of concentrations in the various cells is, of course, only an approximation. It would be justified if, for instance, the contents were perfectly stirred. Alternatively, it may often be justified on the understanding that the 'concentration in the cell' is the concentration at a certain representative point, although the idea of 'concentration of a point' clearly itself raises difficulties. The author believes this approximation is a good one, whatever argument is used to justify it, and it is certainly a convenient one. …"

only holds true if the Molecules are too “large” to be able to pass through the membrane. It is to be expected that the presence of membranes will create interesting interference patterns. For more information on waves of a reaction-diffusion system and their interaction with boundaries see also [Sak02].

- **Channels**: are defined similar to membranes. Again through their position of the center and their dimension. These channels are idealized. Their only function is to bypass the membrane on the defined area.

4.2.2 Operation breakdown

After the last section described the elements of a configuration this section shall describe the operation breakdown of the configuration unit. This operation breakdown has been construed to meet all the requirements as was described in section 4.1.1.

- The first step will be to either create a new configuration – and therefore to specify a new reaction-volume through entering its dimension – or to load an existing configuration from an external file.
- The next step allows the user to specify the simulation mode that the simulation unit should enter. Here the user will be presented with at least three options: diffusion only, the Brusselator reaction-diffusion scheme or selection of a user defined reaction-diffusion scheme. For details on the available simulation modes see also section 4.3.1.
- After these basic steps have been completed the reaction-volume will be displayed. This will allow the user to add additional elements into the reaction-space. In order to do so one of the following options have to be taken:
    - Activate the insertion of the element to be inserted: channel, membrane or species X / Y.
        - Input the parameters for the element. Thereby the user will be given a choice, to either use the mouse to change all parameters or to enter the desired values into a property window. This is in accordance to the analysis of modeling applications as described in section 3.2.1.
        - This will continue until the user either disables the insertion mode, or the file will be saved.
    - Alternatively the user can modify existing elements in the reaction-volume. Therefore an existing element can be selected through clicking on it.
        - That will open the property window for that element. And allow so to move or delete the respective element.
- Once these steps are completed all that is left to do is to save the configuration and / or quit the application. As file format for the configuration data SBML level 2 has been chosen. The reasoning to do so can be found in section: 3.2.2.4.

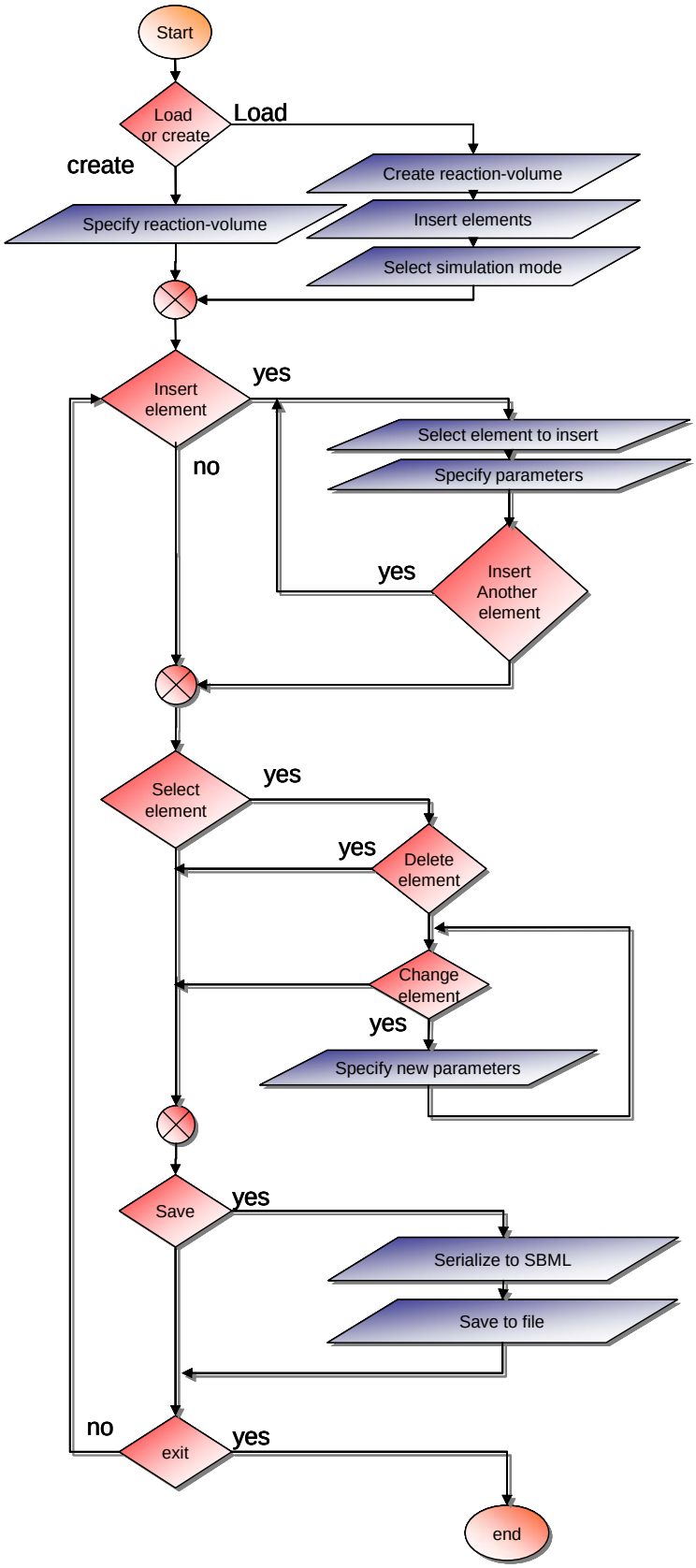


Figure 10: Operation breakdown of the configuration unit

4.3 Simulation

The last section described the concept of creating a configuration. This section will now continue with that configuration and explain the concept how it will be used to simulate the reaction-diffusion system. Thereby it will strongly regard the requirements as defined in section 4.1.2. One requirement was to be able to simulate different reaction-diffusion systems. Therefore several simulation modes have been conceived. These will be described in the next section. After that the operation breakdown for the simulation unit will be given.

4.3.1 Simulation modes

This section will describe in more detail which simulation modes will be supported and what parameters have to be defined for them. Common to all supported simulation modes will be that they will all simulate a reaction-diffusion system as introduced by Alan Turing in [Tur52]. That means they all follow the equations:

$$\begin{aligned} \frac{\partial x}{\partial t} &= F_x(x, y) + D_x \nabla^3 x \\ \frac{\partial y}{\partial t} &= F_y(x, y) + D_y \nabla^3 y \end{aligned} \tag{1.8}$$

(for details see also section 3.3.1). So all the different simulation modes do is provide different equations F_x and F_y (each dependent on the current concentrations of X and Y). In order to provide a large variety of reaction-diffusion systems the aim is to implement a plug-in system. This will allow for the addition of very fast functions (because they will be available in compiled, optimized form). Alternatively a way will be described to use the Systems Biology Workbench to visually design new functions and let them be interpreted.

4.3.1.1 Diffusion

This simulation mode does not include a reaction equation. By setting both functions F_x and F_y equal to zero this mode will result in the simulation of diffusion only. So every configured concentration distribution will eventually diffuse into equilibrium.

The diffusion equations for the two chemicals x and y that are used here is:

$$\begin{aligned} \frac{\partial x}{\partial t} &= rD_x \nabla^3 x \\ \frac{\partial y}{\partial t} &= rD_y \nabla^3 y \end{aligned} \tag{1.9}$$

Here D_x stands for the diffusion rate of chemical x and D_y respectively stands for the diffusion rate of chemical y. The Laplacian operator ∇^3 stands for a measure of the highness of concentration of the chemical at a given location in respect to the concentration of the neighbors. The cubic form hints at the three-dimensional variant.

In this discrete simulation the formulas can be written as:

$$\begin{aligned} \nabla^3 x_{i,j,k} &= D_x \cdot \left(x_{i-1,j,k} + x_{i,j-1,k} + x_{i,j,k-1} + x_{i+1,j,k} + x_{i,j+1,k} + x_{i,j,k+1} - a \cdot x_{i,j,k} \right) \\ \nabla^3 y_{i,j,k} &= D_x \cdot \left(y_{i-1,j,k} + y_{i,j-1,k} + y_{i,j,k-1} + y_{i+1,j,k} + y_{i,j+1,k} + y_{i,j,k+1} - a \cdot y_{i,j,k} \right) \end{aligned} \tag{1.10}$$

Here '*a*' is a multiplier between 0 and 6. This multiplier is initially 6 and gets degreased each time a neighbor is a membrane. This accounts for the fact that no flow through a membrane is allowed.

Although this formula is tailored for the three-dimensional case it holds true for the one- and two-dimensional case as well. The reason for that is that in the two-dimensional case the elements $x_{i,j,k-1}$ and $x_{i,j,k+1}$ again refer to the element $x_{i,j,k}$ and thus the higher negative multiplier gets lowered by two. Similarly in the one-dimensional case $x_{i,j-1,k}$, $x_{i,j+1,k}$, $x_{i,j,k-1}$ and $x_{i,j,k+1}$ refer to $x_{i,j,k}$ and the multiplier will be lowered by four. Thus simulation of one-, two- and three-dimensional reaction-volumes is possible.

4.3.1.2 Brusselator reaction scheme

The second simulation mode will be the Brusselator reaction scheme (First mentioned in [Nic77]). This is a reaction scheme involving two chemicals X and Y engaged in four reactions. In [Tho86] the hypothetical reactions are stated like this:

A → X

B+X → Y + D

2X+Y → 3X

X → E

"Here *A, B, D* and *E* are initial and final products, whose concentrations are imagined to be imposed as constants throughout. All reaction steps are here assumed to be irreversible with rate constants equal to unity."

After stating these reactions [Tho86] continues to derive the rate of production / loss of *X* and *Y*. These rates of changes are needed as return value for the simulation-core. To derive them the same letters are used to denote the concentrations of the chemicals. This leads for the individual reactions to the following result: the rate of production in the first reaction is simply *A*, the rate of loss in the second reaction is the product *BX*, the rate of production in the third equation is X^2Y and finally the rate of loss of *X* in the fourth reaction is *X*.

This leads to the equations:

$$\begin{aligned} rate\ of\ change\ X &= A-(B+1)X + X^2Y \text{ and} \\ rate\ of\ change\ Y &= BX - X^2Y \end{aligned} \tag{1.11}$$

These equations are everything that is calculated by the reaction plug-in. Of course the parameters have to be tended to. '*A*' and '*B*' are substrate parameters. These parameters are to be provided during the configuration phase. Should this simulation mode be selected and these parameters are not provided '*A*' will be set to 2.5 and '*B*' to 5.24. This combination of values is known to lead to interesting patterns.

4.3.1.3 User defined reaction schemes

As already mentioned in the introduction to this section there will be two possibilities to allow for user defined reaction-schemes. The first possibility is to create a plug-in library which can be loaded into the project at runtime. Using this possibility allows for a shorter runtime (since the equations will be available in compiled and optimized form) and thus is preferred. This method will be explained in detail in the implementation section 5.2.1.

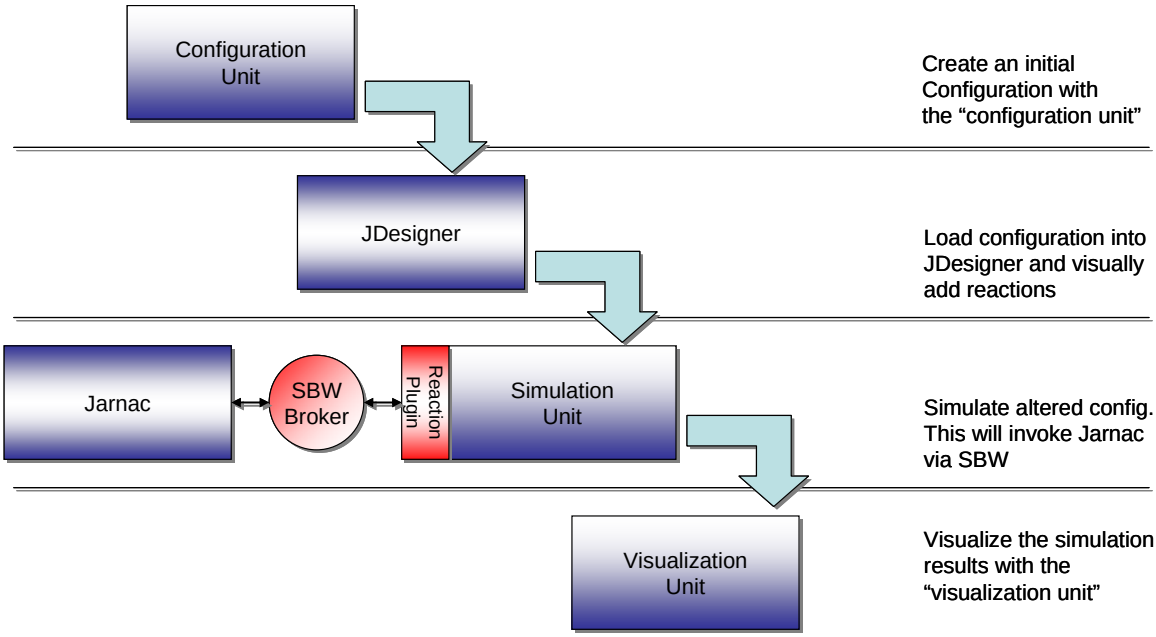


Figure 11: Workflow describing how JDesigner and Jarnac can be used for simulation

The second possibility is to employ the Systems Biology Workbench (as introduced in section 2.3). The main idea here is to create a way to define new reaction-equations visually. This way even users without a background in computer programming could create reaction-schemes tailored to specific initial concentrations of a reaction-volume. Figure 11 describes the actual workflow. First an initial configuration is created. Thereby it is important to select “JarnacReactionPlugin” - during the simulation this will tell the simulation unit that Jarnac is to be evoked in order to compute the reaction-scheme. Other than that the configuration may be created as any other. The second step is to open the created configuration into JDesigner. JDesigner allows defining a reaction network visually. Once the configuration file is loaded into JDesigner the following nodes might already be present:

- The nodes “X” and “Y” represent the species X and Y throughout the reaction-volume. These nodes should be involved in creating the reaction network. That means connections to these nodes should be drawn in order to influence the reaction equations.
- The nodes “channel” and “membrane” on the other side should not be connected.

Once the reaction network is created the user can test it, by letting JDesigner invoke Jarnac to simulate the network. (c.f. section 2.3). Once this file has been saved (as SBML level 2) the configuration is ready to be simulated and visualized.

4.3.2 Operation breakdown

This section will bring together the requirements for the simulation unit and the analysis of section 3.3 through proposing an operation breakdown.

There will be two basic steps for the simulation unit. Step one will be an initialization step. Here the specified configuration file will be loaded and validated. Based on that information the required memory for the reaction-volume will be requested. Next the simulation parameters will be set such as the step size for the integration unit. After that the requested simulation mode will be entered. Finally the initial configuration as specified in the configuration file will be written into the memory reserved for the reaction-volume.

The second step for the simulation unit is a loop consisting of three sub-steps. First a new iteration will be computed. Thereby the simulation-core will first compute the concentration changes for the species X and Y. Afterwards these changes will be applied to the original concentration according to the current integration step. Along with the new iteration some meta-information about this iteration will be computed. These are the minimum-, maximum- and mean-value. These meta-data will later be used by the visualization unit in order to "guess" a threshold for displaying the simulation-data (see also section 4.4.1.1 for a detailed description how this threshold will be calculated). After this computation step the next step will be to either save the current iteration onto the hard drive or pass it on to another application. This loop will be continued until an end is requested. Upon this the simulation unit will stop.

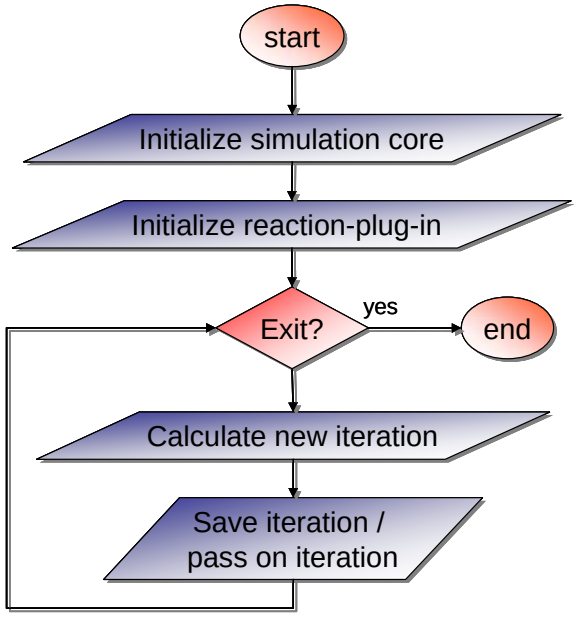


Figure 12: Operation breakdown simulation unit

4.4 Visualization

Before this section will detail the concept for the visualization unit there are two topics to address. The first one will be to take the analysis of common volume-visualization techniques (as described in 3.4.2) into account and generate a list of visualization modes to be supported. A second section will then detail two possible execution modes for the visualization. The essential difference will be that one mode will visualize simulation results "on the fly" and the other mode will display a previously saved simulation run.

4.4.1 Visualization types

As one requirement for the visualization is diversity, there will be several types of volume-visualization algorithms. They generally fall into the following three categories:

- 3D plot,
- Surface fitting algorithms and
- Texture algorithm

As with the simulation modes here too a plug-in architecture will be used. This way it should be rather easy to enhance the existing visualization types at a later stage. The benefit of this will be for example that different visualization libraries might be supported later on. All a plug-in would have to do is to map the existing data-structure into the data structure of the library and then transform the output into the OSG structure. Since all analyzed volume-visualization libraries (for details see 3.4.3) were based on OpenGL this should not be a problem.

Common to all visualization types will be first that transparency can be enabled or disabled and second that the output can be clipped along the three coordinates. Should transparencies be enabled the opaqueness of an object representing a data-point will be determined by that points concentration. This should provide the user the possibility to analyze the output more closely. As expected from a three-dimensional representation rotating and moving the created object will be supported. Finally the user will be provided to simplify the output in three steps: It will be allowed to choose between the display of points only, the display of wire-frames or the display of full surfaces. This way drawing time can be reduced further and thus allow for the display of data-sets that would be to time consuming to display otherwise.

The sub-sections in this chapter will describe visualization types of the main categories further.

4.4.1.1 3D plot

Visualization types in this category basically display each data point above a chosen threshold. This threshold will be either “guessed” by the application based on the minimum-, maximum- and mean-values, or directly entered by the user. For each data point above that threshold one object will be created and colored according to a global palette that assigns a specific color for each concentration. In [Pek01] another method is described to efficiently detect meaningful iso-surfaces. The idea here is to generate a Laplacian-weighted gray value histogram in one pass through the dataset and choosing a maximum value. This is more or less the approach to be taken here in this case. An accumulated histogram with a configurable amount of bins will be calculated. Using that histogram a threshold value will be chosen that reduces the data-points to display by about fifty percent.

There will be two visualization types in this category. A first type will follow the idea of the Cuberille algorithm (see section 3.4.2.3). That means it will create one cube for each data point belonging to the object defined through the threshold. As already mentioned on the section about most common volume-visualization algorithms this implementation will be very slow and thus is intended for smaller simulations only.

A second type will follow the same approach but create only points for each data point. Each of these points will be rendered as a fixed size (otherwise they would not be visible). This simplification makes the display of larger reaction-volumes possible.

The drawback of this type of visualization is that the view gets rather complex if many data-points are to be displayed. The only remedy here would be to clip the display or enable transparency.

4.4.1.2 Surface fitting algorithms

Visualization types in this category will begin to create a surface for the threshold value calculated by the simulation unit. Again this value can be overridden by the user, thus allowing the user to choose which surface to create. For more information on the selection of the threshold value see also the previous section 4.4.1.1.

The first visualization type will be the Marching Cube algorithm and the second type will be the Marching Tetrahedron. (See section 3.4.2.3 and [Bor97] for details on these algorithms.)

Algorithms in this category will be much faster than algorithms in the 3D plot category. This is due to the fact, that fewer objects will be created.

For a general description of surface-fitting algorithms see also chapter 3.4.2.3.

4.4.1.3 Texture algorithm

The idea behind this algorithm is to create one texture for each slice of the reaction-volume in the z-plane. Each pixel of this texture will get a color representing the respective data point, again chosen from the global color palette. The main advantage of this visualization type will be its runtime. Since only one object will be created per slice it will consume a minimum amount of memory and thus will be processed fastest.

The nature of this type of visualization makes it the ideal representation for simulations of two-dimensional reaction-diffusion schemes. For three-dimensional simulations this algorithm still can be insightful in combination with clipping and rotating.

4.4.1.4 User defined

As with the simulation modes (c.f. chapter 4.3.1.3) it will again be easily possible to create new visualization types. An interface will be exposed so that new visualization types can be easily added through loading a new visualization type DLL.

This mechanism will be detailed in the implementation section 5.2.2.

4.4.2 Online vs. Offline visualization

As mentioned in the beginning there will be two supported operation modes for the visualization – online and offline visualization. Before explaining these two operation modes further the reason for introducing these two operation modes will be give. The reason for allowing these two operation modes is the runtime. Large reaction-volumes will take a long time to simulate. Furthermore a large number of iterations might be needed in order to see the formation of reaction-diffusion patterns. These numbers might be as high as 5.000 or even 7.000 iterations. Separating the visualization part from the simulation part may help to solve that problem. This is true especially since it will be possible to change the operation modes. That means that should a user notice that the simulation run that was computed in the offline mode ends prematurely it will be possible to continue this simulation run (either in online or offline mode). The same is true for the other direction. The simulation data from an online visualization can be used in an offline simulation.

Online visualization (i.e. using the visualization unit to load a configuration and visualizing the results while simulating it) will probably be the right type for small to medium reaction-volumes. The advantages of this type lie in the fast generation of images for these kinds of reaction-volumes. Furthermore no additional hard drive space will be consumed. The drawback is that once an iteration is displayed it is not possible to view previous iterations without re-running the simulation. Since the visualization of the data-set will take more time than simulating it there will be the possibility to simulate a user defined number of iterations before visualizing the results. This will speed up the process much.

Offline visualization (i.e. the visualization of data stored from a previous simulation run) is the right type to choose for larger reaction-volumes. For these online visualization would take up too much time. Advantages here are that since the iterations are stored on hard disk it will be possible to go back and forth in time in order to really analyze the dataset. The drawback is obviously the

space needed on the hard drive along with the fact that no feedback is provided whether or not patterns have already formed.

4.4.3 Operation breakdown

This section will now formulate the basic operation breakdown for the visualization part. Thereby the possibility for the two operation modes, offline and online visualization, will be given respect.

The selection of the operation mode will be made by the user. Either the user chooses explicitly to run a simulation, or to display a previous simulation run. After that as an initialization step the original configuration will be loaded. This will be done in order to display the reaction-space, the membranes and the channels. These elements will always be displayed. As further initialization step a fast visualization type will be selected. Further initialization of the simulation-core along with a calculation of the first iteration occurs in the online mode. The last step of the initialization is to display the first iteration (either from a file in the offline mode, or the last computed iteration).

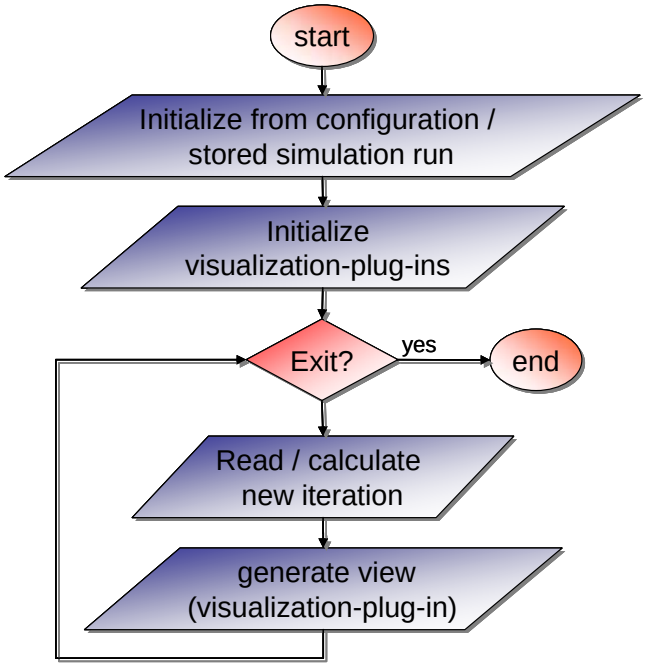


Figure 13: Operation breakdown visualization unit

Now it will be up to the user to perform one of the following actions:

- change render detail (from point, over wire-frame up to rendering of surfaces (default))
- change the visualization type
- clip the displayed reaction-volume
- enable / disable transparency
- enter / leave automatic playback mode
- manually select a different threshold for the visualization
- navigate through the generated display
- quit the application

- save screenshots

Should the user decide to enter the automatic playback mode a new iteration will be loaded / computed according to the current visualization step-size. This allows for example in the online mode to calculate a larger number of iterations before the last iteration will be displayed. In the offline mode this will help to skip some iterations.

4.5 Conclusion

The aim of this section is to wrap up what has been achieved with this conception. In what way was the conception done and why so? It will show similarities between the separately described parts.

Each of the three parts, configuration, simulation and visualization has first been analyzed in order to compile requirements that are essential for the individual tasks. After describing unique elements for each part (the elements of the configuration, simulation and visualization modes) all sections contained a basic operation breakdown. This operation breakdown described the individual points that have been given critical importance to a successful, usable application. Thus the points mentioned there are points to be implemented.

To conclude this chapter, some final remarks on the separation. Separating the configuration unit from simulation and visualization brings along the following benefits. First of all it makes it easily possible to create the configuration with third party application. Though at the moment modeling applications for reaction-volumes that regard spatial information are still hard to find, they may be developed in the near future. The idea of using JDesigner to enhance a configuration, created with the configuration unit, displays one way to widen the range of configuring at least the reaction-networks. Secondly the separation allows the user to save time. It allowed to create both visualization modes, online and offline, and with it combining or separating the simulation and visualization unit. Furthermore the workflow for altering a running simulation by changing to a new configuration is much more easily possible with the separated approach. Finally, since the simulation-core will be written with strong regard for platform independence this will open up the possibility to run simulations on a more powerful system than the desktop-workstations targeted for the visualization. A last point supporting the separation into individual parts is that it allows for a certain kind of specialization. While an expert biologist is certainly needed to create the initial concentration, an expert for simulation could optimize the model to simulate and finally an expert in volume-visualization might work on the best display of the generated data. This will be possible since the three parts are separated. The biologist needs to pass on the configuration, while the simulation expert passes on the simulation-data. This enables the volume-visualization expert to find a fitting visualization.

The process of implementation along with its problems and limitations will be discussed in the next chapter.

5 Implementation

Based on the analysis of chapter three and the conception of chapter four this chapter will first describe the data-structures used. Following that the architecture of the system will be described further. A final point will talk about difficulties that were encountered along with limitations of the solutions that could be found.

As already stated in the last chapter, the implementation will include all points of the conception. This will provide for an easily usable, extendable simulation and visualization application.

5.1 Data-structures

This section will begin with explaining the data-structure used to hold the configuration data. Next the internal data structure that holds the simulation data will be discussed.

5.1.1 Configuration data-structure

The configuration data-structure (in the implementation named Configuration) holds all data that is necessary to run a simulation. It is being used by the configuration unit, where it will be created, by the simulation unit for setting up the initial concentrations and initialization of the simulation-core and by the visualization unit in order to represent the reaction-volume along with possibly existing membranes and channels. This subchapter will explain this structure in more detail.

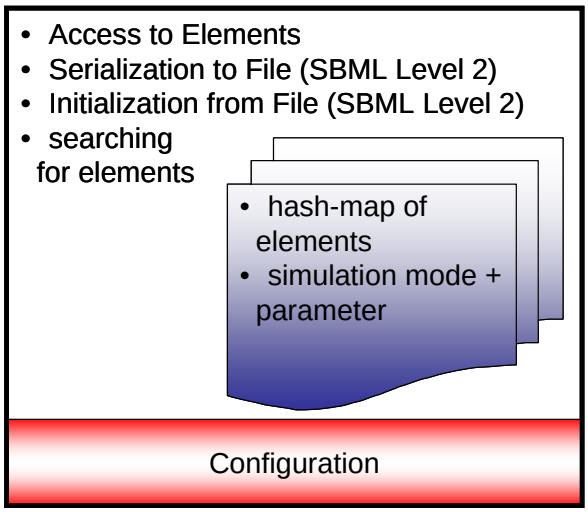


Figure 14: Elements of the configuration data structure

The elements of the configuration fall into three categories. The most basic of these categories is the data needed to specify the reaction-volume. In this first implementation this is just the dimension of the reaction-volume. The second category represents data needed for the specification of the simulation mode. In detail these are a reference to the selected reaction-plug-in and the configuration of that plug-in. The third category is represented by all elements that have been inserted by the user. Here a hash-map based on the position of the respective element as key and a reference to the element as value has been chosen. Special classes exist for the possible elements, each

derived from a common class "`Element`". In order to retrieve the respective element at a later time the C++ run time type information (RTTI) will be used to obtain the object.

Accompanying these data are support functions. Apart from basic "get-/set-methods" and "insert-/delete-methods" for the hash-map, the most important functions are probably the ones used for serialization or de-serialization. As already stated in the analysis of possible file formats in section 3.2.2 the file format SBML (level 2) will be used to store the data. Section 3.2.2 also provided information about the way the configuration data can be applied to the model-data represented by the SBML document. In order to implement an export of the configuration it is necessary to save some information as SBML annotations. The reason for this is that SBML currently does not support spatial-data for the species or defined compartments. Thus a XML name space was defined after [Van02] that specified the needed elements. These definitions are displayed in the appendix: "C - Schema Definition of SBML Annotations".

5.1.2 Simulation data-structure

The aim of the simulation data-structure (in the actual implementation named DataHandler) is to provide access to all the data stored during one iteration. The reason for storing the data in this class and not along with the simulation unit was the following. In the online operation mode of the visualization unit the simulation-data will be displayed after each iteration. Each iteration might contain a huge number of data. For each element of the reaction-volume four double values (concentration for species X and Y, along with the rate of change for X and Y) will be needed for the simulation and two double values (current concentration for species X and Y) will be needed for the visualization. To illustrate that with an example: for the simulation of a grid with dimension 128x128x128 this would lead to 64MB of memory needed to simulate such a grid. Of these data 32MB are needed for the visualization. Copying these data would on the one side slow down the visualization process and on the other side occupy more memory. This is where the DataHandler into play. Apart from storing the data for the concentrations of species X and Y along with their rates of change it also provides an interface so that both applications can access these data.

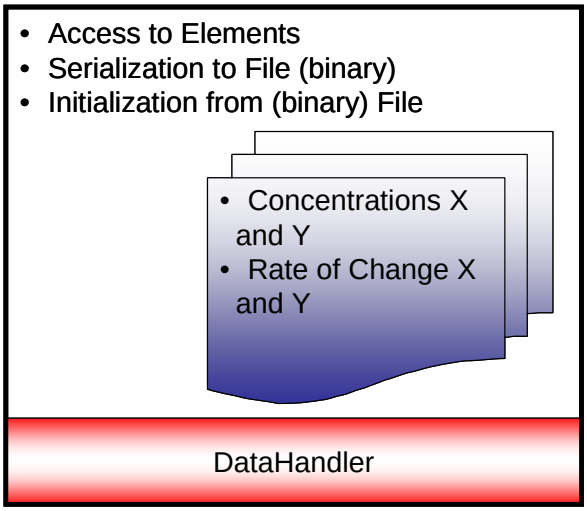


Figure 15: Elements of the DataHandler data structure

5.2 Plug-in system

Both the simulation unit and the visualization unit are written in a way to easily extend existing capabilities. Therefore a plug-in system was conceived that will be described here. First the basic architecture will be detailed and following that it will be specified how it will be used for the simulation and visualization types.

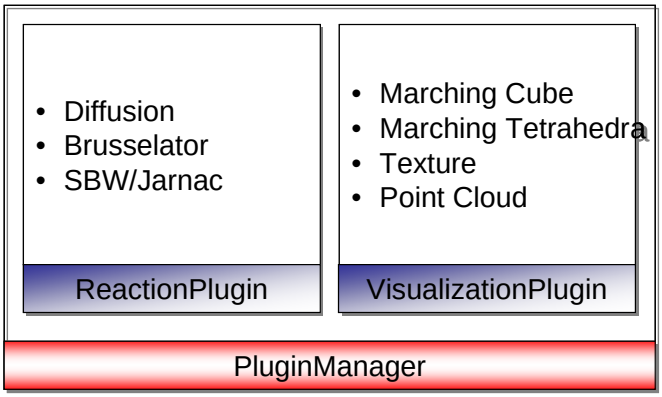


Figure 16: Elements of the PluginManager

There will be one manager for both types of plug-ins. An instance of this manager will be created at the program start. Each plug-in library will contain one registration method that will be called upon loading. This method will register each and every plug-in of the library. Registering in this case means that a prototype instance of the respective will be created and given to the plug-in manager.

Upon creating the user interface for the configuration or visualization unit, or during loading of a configuration the plug-in manager will be asked for the available plug-ins. Should they be needed the plug-in manager will provide an instance of these plug-ins.

5.2.1 Reaction plug-ins

Reaction plug-ins determine the simulation mode that will be entered. As already described in section 4.3.1 the reaction plug-ins specify the functions $F_x(x,y)$ and $F_y(x,y)$ in the basic reaction-diffusion scheme:

$$\begin{aligned}\frac{\partial x}{\partial t} &= F_x(x,y) + D_x\nabla^3 x \\ \frac{\partial y}{\partial t} &= F_y(x,y) + D_y\nabla^3 y\end{aligned} \tag{1.12}$$

Each reaction plug-in has to be inherited from the abstract class: "`ReactionPlugin.h`". This class definition represents the interface through which all calls will be made to this plug-in. In order to create a new reaction plug-in the following members need to be implemented and exported:

- `void initialize(std::string sConfigString)`: Although the prototype of each plug-in will be stored in the plug-in manager, the initialization should happen when this function will be called, as opposed to initializing the plug-in in the constructor. The reason for this is that on the one side specific initializations may take a long time and on the other side the plug-in may take certain parameters which will not be provided by the default constructor.

As parameter this initialization function will be called with the configuration string as specified during the configuration. The configuration will display the default parameter as provided by the next function to implement.

If the parameters are in the standard format, meaning "`<name>=<value>;`" then the base class will parse these parameters and they can be obtained by calling:

`void parseParameters(std::string sConfig);` and

`std::string getParameter (std::string sName);`

- `std::string getDefaultParameter()`: This function should provide the default parameter for the reaction plug-in. Since the parameter has to be returned as a string the following format should be used to construct that string:

  `<ParameterName1>=<value1>; <ParameterName2>=<value2>;` …

  An example would be "`A=2.5; B=5.24;`" this parameter is used by the Brusselator reaction plug-in.

- `const char* className() const:` this function provides a class name for the reaction-plug-in. This class name will later be displayed by the configuration unit in order to allow the user to make a suitable selection.

- `double getChangeX(double dConcentrationX, double dConcentrationY)` and

  `double getChangeY(double dConcentrationX, double dConcentrationY)`:

  These are the main functions to implement. These are the functions that will be called over and over by the simulation-core. Thus it is vital that these functions are written in high regard for performance and memory consciousness.

5.2.2 Visualization plug-ins

The visualization plug-ins provide a possibility for diversity in the display of the simulation results. Each of the plug-ins derived from "`VisualizationPlugin.h`" will benefit from a couple of support functions, that will help to provide information about the mapping from concentration value to the color that concentration is suggested to be assigned. Furthermore the access to the dataset will be provided along with information about mean-, minimum- and maximum value in the data set. Of course the threshold selected by the user will be provided as well along with the information whether to support transparency or not.

The functions that have to be implemented when deriving from the abstract class "`VisualizationPlugin.h`" are the following:

- `const char* className() const`: this function will provide the information to the plug-in-manager with which this plug-in is to be found in a later stage. Also this information will be displayed to the user in order to help him to make a selection.

- `osg::ref_ptr<osg::Node> generateNode(double dThreshold, DataHandler *oHandler)`: For the visualization plug-ins this is the key function to implement. Along with a reference to the `DataHandler` (containing all simulation data as described in 5.1.2) the threshold value will be given as argument to this function. It is now com-

pletely up to the plug-in to process the provided information and generate an `osg::Node` (the base object for any scene graph element) containing all the visualization information.

5.3 Design of the system

This section will explain the design of the system based on the classes created after the conception of the last chapter. To that aim first the complete structure of the system will be detailed before the individual parts for configuration, simulation and visualization are described further.

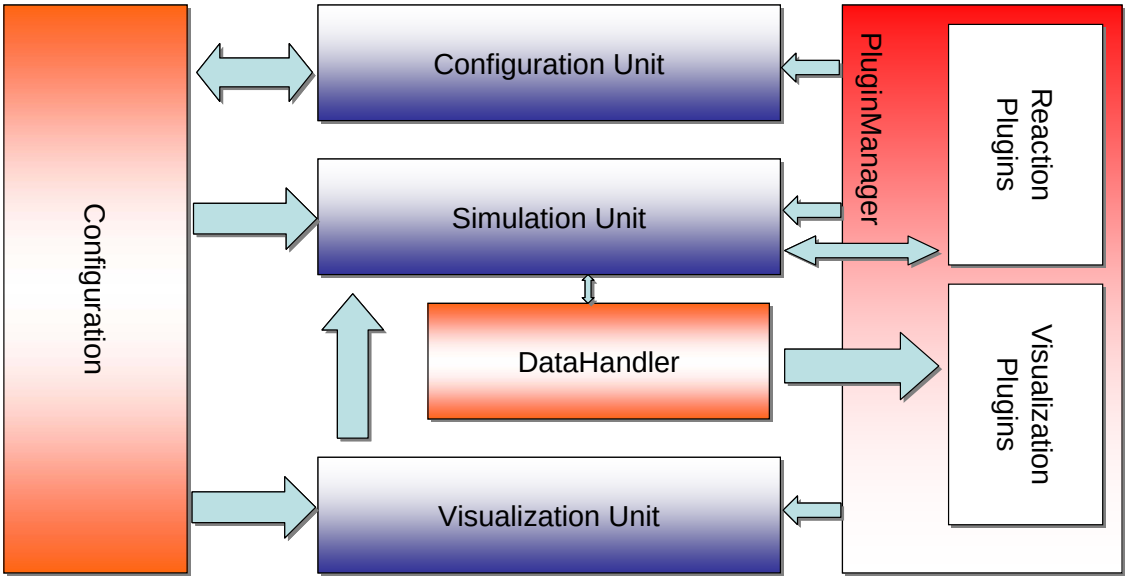


Figure 17: Overview of the systems design

Figure 17 displays the schema that details how the individual components of the developed application are connected. As expected all the components are strongly tied together. The configuration of the system plays a vital role for the simulation as well as for the visualization. On the one side it is needed in order to initialize the simulation-core and on the other side basic information is taken out of this configuration in order to visualize the reaction-space. Another vital component is presented by the plug-in system. This provides the user to create configurations for different reaction-diffusion systems, which are then processed by the simulator. For the visualization of these simulation results, accessed through the data handler, the plug-in manager provides several visualization types. For more information see also section 5.2.

In the online visualization mode the visualization unit works closely together with the simulation unit. They both share a common dataset provided by the `DataHandler`. This was already described in section 5.1.2.

5.3.1 Configuration

This sub section will describe the configuration unit that was implemented in order to fulfill the requirements as stated in sections 4.1.1 and 4.2. The focus thereby lies on the description of the program logic. A description of the graphical user interface and with that the integration of the used three-dimensional renderer can be found in the next section (5.4). Figure 18 lists the essential parts for the configuration unit.

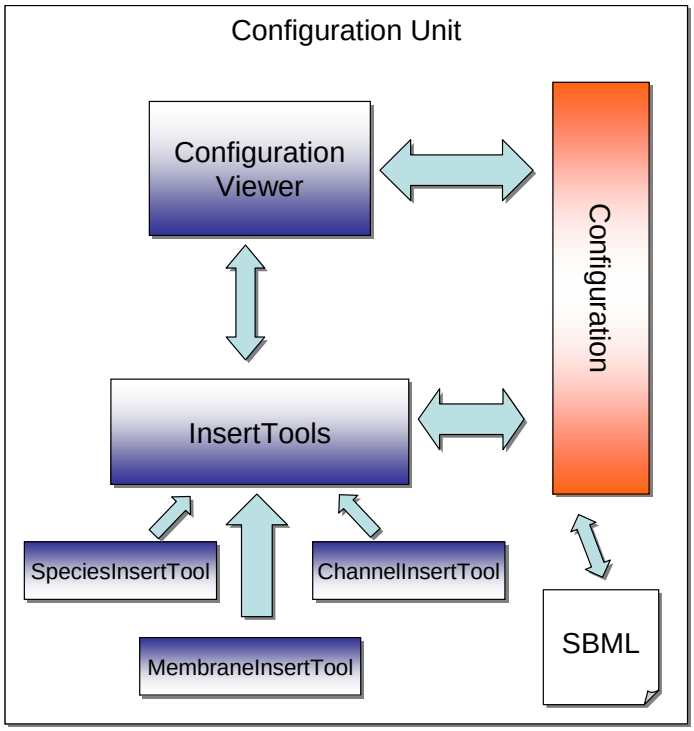


Figure 18: Configuration unit schema

The most interesting parts of the configuration unit are certainly the following:

- how are the data manipulated in the configuration data structure (as mentioned in 5.1.1)
- and how are elements added to the configuration (such as channels, membranes and species).

The main component that makes these tasks possible is the "`ConfigurationViewer`" it allows to display configurations (as provided by the data structure), to navigate in the created reaction-volume and to add or modify existing elements and finally to save the created or modified configuration.

For the insertion of the specific elements insertion tools were created. They all derive from a common base class "`InsertTool.h`" that provides the basic interface for the `ConfigurationViewer`. Features provided by the base class contain functions to enable or disable an insertion tool, to move the insertion tool and constraint checking. Constraint checking is used so that no element can be inserted outside of the reaction-volume. Finally the base class provides an interface for keyboard input. Functions to implement by each of the derived classes (for this case rudimentary implementations were made for the tools: `ChannelInsertTool`, `MembraneInsertTool` and `SpeciesInsertTool`) are functions to create an element based on the current setup of the tool, to provide a graphical representation for that element (or any given element of the supported type), to provide a graphical representation for the tool itself and to handle keyboard input. These insertion tools receive keyboard information from the viewer. This is used for example by the channel- or membrane insertion tool in order to specify the dimension of the object to insert. Via pressing the left or right key the width will be restricted or widened (for the height the up-/down keys are used and for the depth the PGUP/PGDOWN keys). So the insertion process works like this, first the user enables the respective insertion tool, then the tool is moved via the mouse to the desired position. Now the user can specify optional parameters, like the dimension for a channel or membrane or the concentration for the specie. Finally the object will be created by pressing the Space-key. This will continue until the user disables the insertion mode.

Selection of elements was implemented to allow the user to modify/delete already placed elements. Therefore the user would use the mouse to click on the desired object. This will invoke a picking routine to test whether any element was hit by the click on that position. Should an object be hit, it will be highlighted as selected. Selected elements can then be deleted or moved around.

5.3.2 Simulation

The implementation of the simulation unit was done with the thought in mind to create a platform independent application. Furthermore the simulation unit had to be able to be integrated to the visualization unit in order to allow for the online operation mode of the visualization section. As stand alone application it will represent a command line tool. The stand alone application was created to process the simulations on different machines and later use the created simulation-data to perform an offline visualization. The idea of this process, first simulate a certain number of iteration steps and later on visualizing the created data, poses the need for two different execution modes of the simulation unit. First of all the simulation unit should be able to simulate a configuration file and store these information. The second execution mode has to be able to continue a previously run simulation. This will be necessary since a previous simulation might end at a point where reaction-diffusion patterns are not yet “stable” (or have not yet begun to form). For the online visualization mode, where usually no data will be written onto the hard drive, this mode allows to use the simulation-data of a previous simulation run as start point for an online visualization. Finally a function was implemented that allows changing the configuration of a running simulation to another configuration with the same dimensions of the reaction-volume. This allows altering a running simulation. This feature will be accessible in online visualization only.

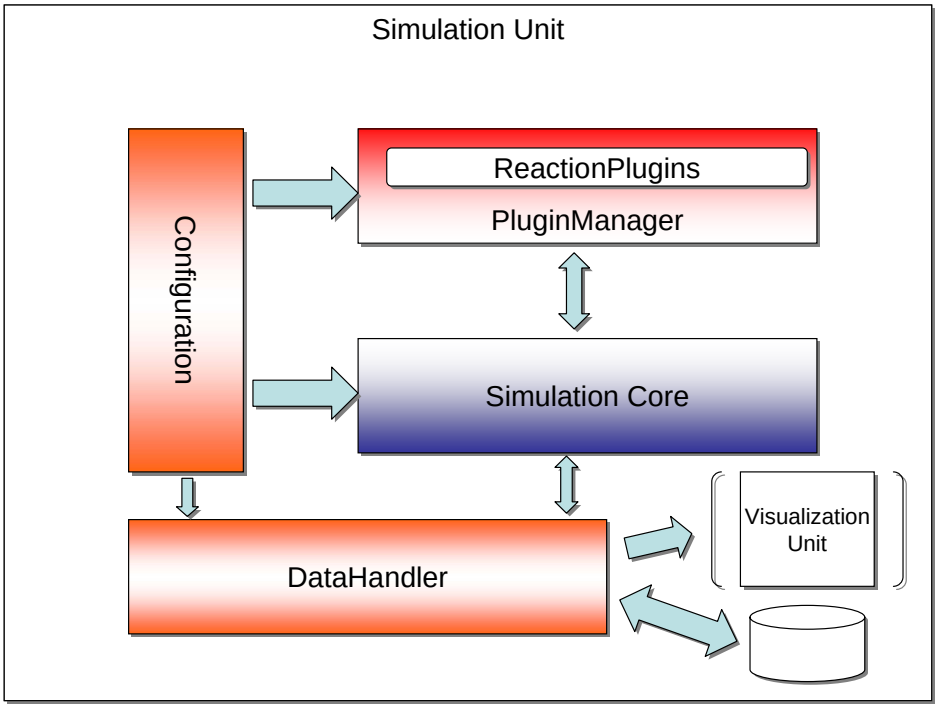


Figure 19: Simulation unit schema

The Figure 19 explains the operation mode and collaboration with the other components. First a configuration object will be generated from a filename or from the information of a previous simulation run. This configuration will then be used to initialize the reaction plug-in to be used, the simulation-core and of course the reaction-volume represented by the DataHandler (see also 5.1.2).

In case that a previous simulation run will be continued the stored simulation-data of the last iteration will be taken as initial concentration distribution for the reaction-volume. Otherwise the elements of the configuration file will be taken to set up an initial distribution for the reaction-volume.

After the initialization the simulation-core will perform the following steps. In the standalone application it will calculate an iteration (based on Turing's equations as stated in sections 3.3.1, 4.3.1 and 5.2.1). As integration unit (as analyzed in section 3.3.2) a simple Euler method has been chosen after all. This has been done in reflection on the runtime. Should the Euler method fail, (highly unlikely for the specified concentration range 0.0-6.0), the simulation will interrupt the calculation of the current iteration. The user will be notified of the fact and informed that by setting a different (lower) integration step size this error will be resolved. A failure in the integration would otherwise result in negative concentrations, which would falsify all following iterations. Once the computation is complete the simulation-data will be saved onto the hard drive. These steps will be continued until the user breaks the execution.

Should the simulation unit be called from the visualization unit only two steps will change. First instead of saving the simulation-data onto the hard drive they will be accessed through the `DataHandler` and passed on to the selected visualization plug-ins. The other thing that changes is that it will be possible to alter the running simulation by selecting another configuration file with the same reaction-volume dimension. This will be detailed in the next section.

5.3.3 Visualization

The implementation of the visualization unit will support two execution modes. These modes, as described in the conception, are titled online and offline visualization mode. For the user this will allow two approaches to visualization. First of all it is possible to let the simulator process a configuration without the need to attend the computer. These data will then just be saved to the hard drive. At a later time the user can then choose to visualize the simulation-data. The other visualization mode, titled online visualization, will take a configuration and use it to simulate one iteration at a time and then display it. In order to change from one simulation mode to the other the following steps will have to be performed by a user:

- online- to offline-visualization: All the user has to do to perform this transition, is save the current simulation results from the visualization unit. And call the simulation unit with these results as parameter.
- offline- to online-visualization: Here the user will perform an offline visualization up to a certain iteration and then stop it. Now instead of choosing to display these iterations the user will use the visualization unit to "continue a previous" iteration, which will take the last iteration as a start point for the online visualization.

Should another configuration be opened, although an online-visualization is currently in progress, it will be possible to alter the current simulation by adding the data of this new configuration file. In order to be able to do this the new configuration has to have the same size of the reaction-volume. What will happen internally is the following: from the old configuration all membranes and channels will be deleted from the reaction-volume and only the concentration of species X and Y along with their respective rate of change will remain. To these the data specified in the new configuration file will be added. Using this feature it will also be possible to change the simulation mode by selecting a different reaction plug-in for the new configuration. This concept will be vital

for the understanding and creation of complex patterns. These patterns might not form from one initial concentration. Rather they might be gradually evolved.

A common feature for both online- and offline-visualization will be a “playback” mode. This mode will display one iteration, either reading it from a file or simulating it in the background, and then make a first visualization. According to the selected step-size, that will decide the number of files to skip or the number of iterations to compute, the next iteration will be displayed. This step size is set initially to one. The user will be able to change this step-size interactively by pressing the *plus* or *minus* key on the keyboard. Alternatively a menu will provide often used step-sizes. This “playback” mode can be entered upon pressing the *space* key or pressing a button, and can be cancelled in the same way. Optionally can be activated that a screenshot is made upon each iteration and stored onto hard-drive. Canceling the playback mode will result in displaying the last calculated iteration in the online-mode.

At any time the user will be able to decide whether to enable or disable the transparency mode (by pressing “*T*”) or to choose a different render mode (by pressing “*W*”). By default transparency is enabled. The default rendering mode is to render surfaces. Pressing “*W*” will result in a cycle between point rendering only, wire-frame rendering and surface-rendering. It is also possible to generate a screenshot anytime by pressing “*S*” or “*PrintScreen*”. Also the user will be able to change the threshold for which the visualization-plug-in will calculate the view. Furthermore it is possible to clip the calculated view along each coordinate-axis. Clipping the reaction-volume along with the different opaqueness might be the only possibility to gather all desired information from the reaction volume. Otherwise vital information might be “hidden” by concentration on a concentration-slice closer to the user. Finally it will be possible to change the visualization type by selecting a different visualization plug-in anytime.

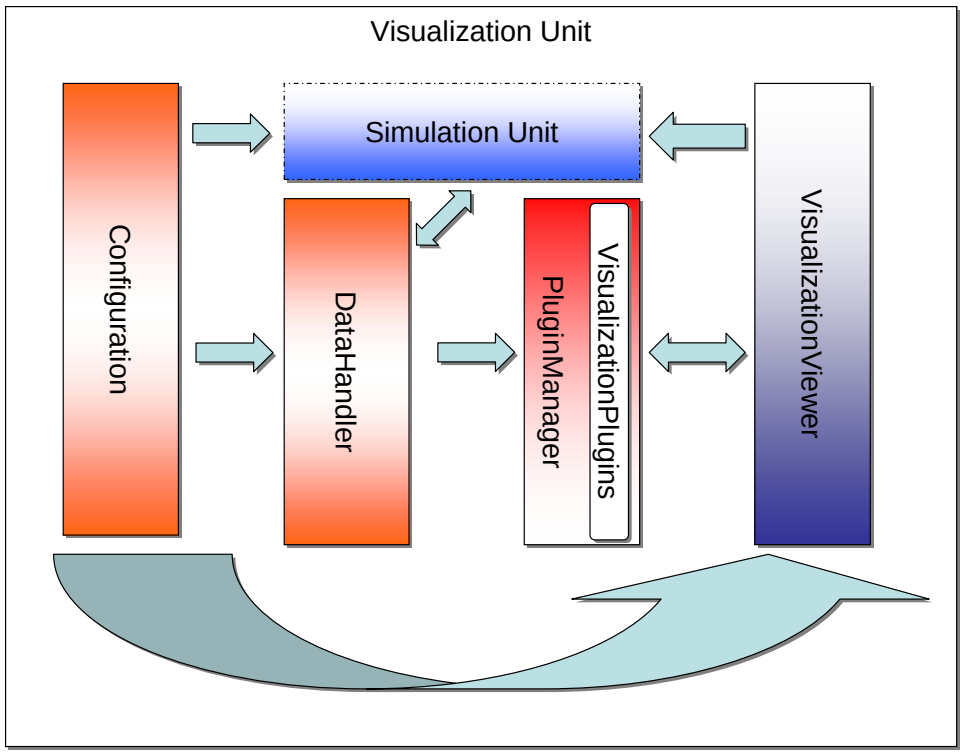


Figure 20: Visualization unit schema

Figure 20 displays a scheme how the visualization unit works together with other created components. After the user decides what should be visualized the `VisualizationViewer` will either receive a filename, in case of online-visualization, or the location of the stored simulation-data in

the case of offline-visualization. A collection of stored simulation-data is always accompanied by an information file. This will tell the `VisualizationViewer` from which configuration this simulation run was created. In a first step a `Configuration` object (described in 5.1.1) will be created. This object will be used to visualize information such as the dimension of the reaction-volume along with possibly existing membranes and channels. In case of an online-visualization this configuration will be passed on to the simulator, in order to initialize it. The last step of the initialization phase is then to calculate an initial iteration in case of online-visualization or to load a first stored iteration into the `DataHandler`. Then a default visualization-plug-in will be selected and the iteration will be visualized by the plug-in through access of the `DataHandler`.

5.4 Graphical user interface

Both, the application implemented for the configuration unit as well the application implemented for the visualization unit, have been implemented with regard to a rich graphical user interface that has the aim to support the user during each program step. Thus both application support standard windowing features such as recent-files (a dynamic file menu listing the last files opened sorted by last access), drag-and-drop (using the mouse to drop a file onto the application in order to open it), quick-tips (an information box that appears if the user hovers with the mouse over a GUI element), dockable windows (windows that can be un-docked from the application and be positioned by the user) as well as menu-bars, toolbars and status-bars.

This demanded for a highly capable windowing toolkit. Thus a first step towards implementing the graphical user interface was to compare commonly used windowing toolkits these are Microsoft WinForms, wxWindow, Gtk+, the FOX toolkit, and the Qt library. The only restriction applied to the author was not to use Microsoft Foundation Classes (MFC) a highly capable, proprietary windowing toolkit developed by Microsoft. Microsoft WinForms a new windowing toolkit developed by Microsoft and just recently (since VisualStudio 2003) available for C++ (.NET) could not be brought to work together with OpenSceneGraph, the chosen rendering API and thus was not pursued any further. wxWindow and Gtk+ though widely used throughout the developer community were not selected due to bad experiences of the author in a prior project[19]. The wxWindow toolkit created problems in a multithreaded application. And Gtk+ though very popular in LINUX environments does not comply to standard concepts on MS Windows platforms (standard controls did not provide the same functionality as their Windows counterparts such as Clipboard support). The FOX windowing toolkit developed by Joren van der Zijp (c.f. [Van,04]) since 1997 as open source project was another possibility. FOX also is platform independent, C++ based, and supported OpenGL (and thus could be made to support OpenSceneGraph). Nonetheless the FOX toolkit was not used since the documentation of this project is still lacking in vital parts and necessary information can only be obtained through mailing lists or source code studies. Furthermore as expected from open source work FOX is still work in progress and not yet complete. Qt as commercial project was developed by Trolltech (c.f. [Qt04] and [Bla04]) as a C++ toolkit for multiplatform GUI and application development. It follows an easy to understand concept of signals and slots, is very well documented and easy to extend. These factors together with the fact that a free non-commercial version of Qt (version 3.2.1) came along with [Bla04] made Qt the windowing toolkit of choice for implementing the GUI for the configuration unit and visualization unit. The only

[19] Gtk+ as well as wxWindows were temporary selected as windowing toolkits for the AMIRE project (c.f. [AMI04]). In the long run it was decided to use MFC over these windowing toolkits because of their shortcomings in MS Windows environments.

restriction of the non commercial version is that the string “[non-commercial]” will always be included in the title bar of the application along with a system menu “About Qt”.

This section will continue to describe the implemented graphical user interface for the configuration unit and visualization unit.

5.4.1 Configuration unit

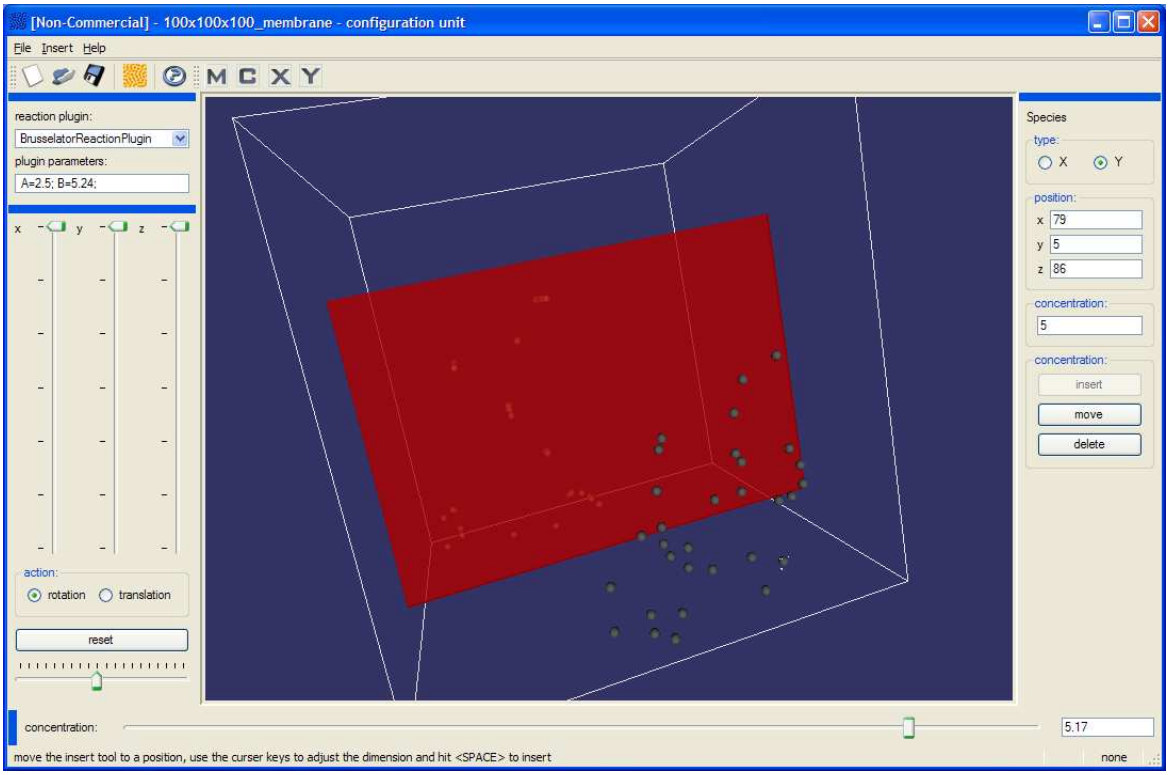


Figure 21: Main screen configuration unit

Figure 21 shows the main screen of the configuration application. This subsection will continue to describe the main features:

- **title bar:** Along with the program name the title bar will always display the currently opened filename or “untitled” should the current configuration be a new one.
- **menu bar:** The menu bar contains two menus. First the file menu, with the standard features Open, Save, SaveAs, recent files and Exit. The second menu provides access to the insertion tools. Here the insertion mode can be set for the insertion of species X and Y, channels and membranes. Of course it will be possible to disable the insertion modes as well. To provide the user with a feedback which insertion mode has been selected these entries will be displayed sunken if activated.
- **tool-bar:** The toolbar provides the user with fast access to the most common used of these functions. It is also possible to start the visualization unit from the toolbar. This will pass the currently edited configuration to the visualization unit.
- **OpenSceneGraph Viewer:** The main control for the application is represented by the OSG Viewer. This object allows the three-dimensional representation of the configuration. Basic features provided by this object are navigating through the scene by using the mouse. By left-clicking and moving the mouse or the mouse-wheel the scene will be translated around the

corresponding coordinate-axis. On right-clicking and moving the mouse or the mouse-wheel the scene will be rotated along the corresponding coordinate-axis.

If an insertion mode is activated the OSG Viewer will display the corresponding tool. (a sphere for the `SpeciesInsertTool`, a blue cube for the `MembraneInsertTool` and a green cube for the `ChannelInsertTool`). This allows the user to drag the insertion object through the reaction-volume. It will not be possible to drag the tool outside the reaction-volume. Once the desired position is reached and space is pressed the respective element will be inserted on that position. (see also section 5.3.1)

Should the insertion mode be disabled, left-clicking upon an element in the scene will result in selecting it. It will then be possible to drag this element to a different position or to delete it.

- **navigation dockwindow:** In order to provide the user with the possibility to easily navigate through the scene this dockwindow has been created. It allows the user to reset, rotate or translate the view. The speed of the mouse wheel can also be configured by this window. As explained in the introduction to this chapter dockwindows can be undocked from the application and placed wherever the user likes. It is also possible to hide this window by disabling it from the popup menu that appears when right-clicking on it. This is true for all dockwindows.
- **plug-in dockwindow:** The plug-in dockwindow allows the user to specify the reaction plug-in to be used along with its supported parameters.
- **concentration dockwindow:** This dockwindow provides the user an easy interface for specifying the concentration for the next specie element to insert. The concentration will be chosen either by a slider, or by directly input in the textbox. Should the slider be moved the range is restricted to the range 0.0-6.0. This range was chosen based on results achieved in many simulation runs.
- **insertion dockwindow:** On program start this dockwindow is not visible. It will be visible only if the insertion mode is activated or the user selects an element. Then according the insertion mode activated it allows to input the parameters for the respective element (dimension and position for channels and membranes and position and concentration for species). Upon selecting an existing element the dockwindow will display the parameters of the selected element. The difference between these two activation modes is that for active insertion mode this window allows only for insertion of elements whereas in selection mode it will only allow to move or delete an element.
- **status bar:** The status bar is divided into three parts. The first part will display information regarding the current operation. The second part will inform the user if the configuration was modified. The third part will display the currently active insertion mode and cycle through the stages “none”, “channel”, “membrane”, “species x” and “species y”.

Upon closing the application, or the loading or creation of a configuration a check will be performed whether the current simulation was modified. If so the user will be prompted to save the changes, to cancel the operation or to continue the operation without saving.

Information like recent files, the current window position and size will be saved upon closing the program. Thus the window will appear on the same position and with the same dimension upon starting the program anew. Since Qt functions are used to store this information the save location is dependent on the platform. On Windows systems these settings will be stored in the windows registry whereas on other platforms they will be written in a configuration file (not to be mistaken with the configuration files created by this application).

5.4.2 Visualization unit

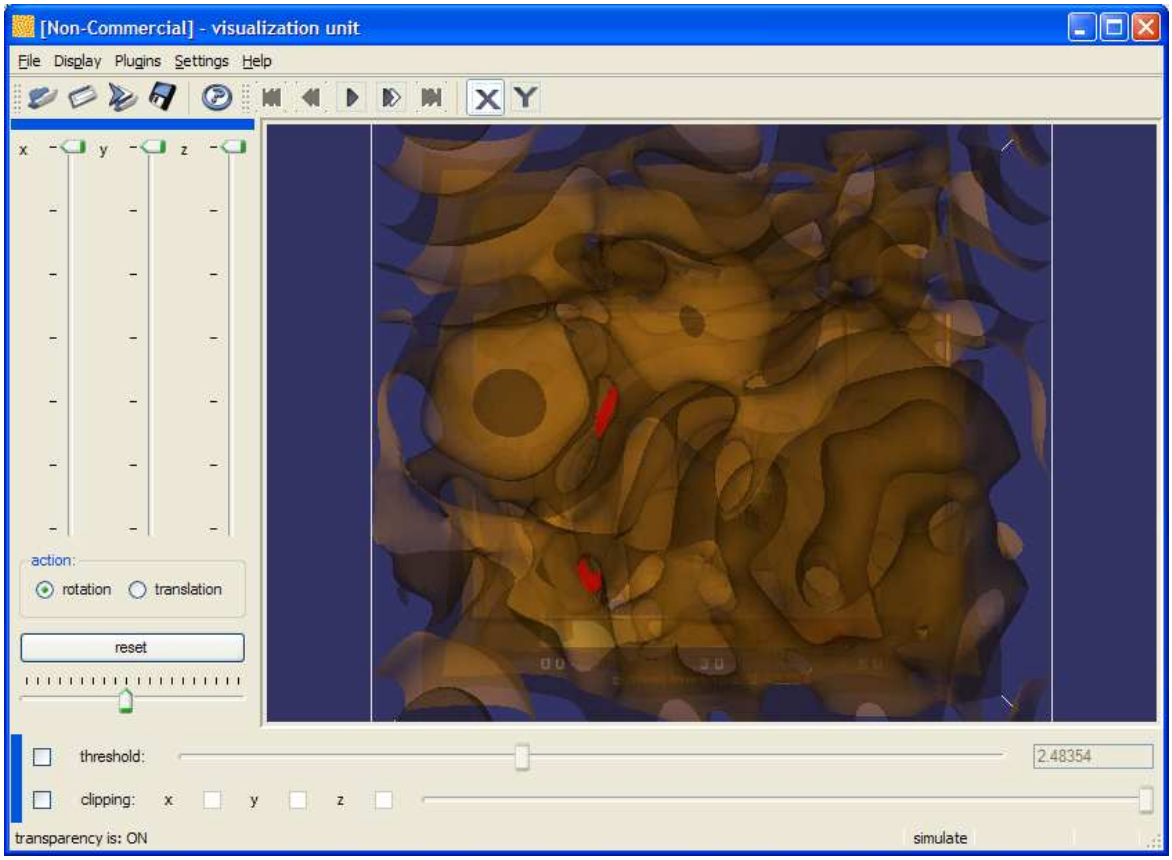


Figure 22: Main screen of the visualization unit

In Figure 22 the main screen for the visualization application is displayed. It consists of the following parts:

- **title bar:** Similar to the configuration application the title bar will display the opened filename or directory.
- **menu bar:** The visualization application contains the following menus. The File menu provides the user with the following actions:
    - "Open Simulation" clicking upon this element will open a dialog where the user will be prompted to select the directory containing the simulation-data to visualize.
    - "Run Simulation" this will allow the user to select a configuration file that will be simulated and visualized.
    - "Save Simulation" this element allows the user to save an online visualization.
    - "Continue Simulation" this allows the user to continue a previous simulation run, either saved using the last option, or saved by an offline visualization.

  The second menu "Display" allows for the selection of the concentration to display. Either the concentration of species X will be displayed or the concentrations of species Y.

  The menu "Plug-ins" will provide the user with the possibility to select the visualization type for the current iteration. Figure 24 shows all available visualization plug-ins as they visualize a Brusselator-simulation with a reaction-volume of 50x50x50. How this configuration was created is explained in the appendix B - Examples.

Finally the menu “Settings” is there to allow quick selections of different visualization step sizes. These influence what happens when a different iteration should be displayed. In the online visualization mode a new visualization occurs only after as many iterations were computed as are specified with the step size. In the offline visualization mode when pressing the back and next button (see playback toolbar) as many iterations will be skipped as defined with step size. Apart from specifying the step size this menu also allows to change the settings of the simulation unit (see also dialog simulator settings). Changing these settings will only be available for online visualization.

- **tool bars:** The visualization application has two toolbars. The first one is there to allow fast access to selected actions from the menu bar. The second toolbar allows control which iteration will be displayed next. For the offline-visualization this allows to go to the first, previous, next or last iteration. For the online-visualization it is only possible to go to the next iteration. This is due to the fact that no iteration will be saved onto the hard-drive and only the current iteration is in memory. The “next” or “previous” iteration is dependent on the selected step-size. Selecting the “play” button will result in activating the automated playback mode. Here the “next” iteration will be displayed shortly after the current iteration is displayed.

- **navigation dockwindow:** The navigation dockwindow provides the same functionality as the one described in the explanations to the configuration unit. In fact it is the same control. (see 5.4.1).

- **tool dockwindow:** The tool dockwindow allows the user to specify a different threshold than the one calculated by the simulator based on the mean value. With that function it is possible to display different iso-surfaces. To change the threshold the user activates the first checkbox on the tool dockwindow. This will enable the slider and the textbox in the first row. There the new threshold can be selected. Apart from selecting a different threshold this dockwindow also provides the possibility to clip the displayed iteration. Therefore the checkbox representing the coordinate-axis to clip to has to be selected and then a percentage set with the slider. Figure 23 demonstrates how a different threshold will influence the created visualization.

- **OpenSceneGraph viewer:** The OSG viewer for the visualization application provides only navigational features apart from the display of the reaction-volume along with a representation of the current iteration. Other than that it is similar to the one described in section 5.4.1. Again left-clicking and moving the mouse or the mouse-wheel will result in translating the scene around the corresponding coordinate-axis. On right-clicking and moving the mouse or the mouse-wheel the scene will be rotated along the corresponding coordinate-axis.

- **status bar:** The status bar is divided into four parts. The first part is there to display information about current executed actions. The second part indicates whether the online visualization has been chosen (“simulate”) or whether offline-visualization is active (“replay”). The third part informs which iteration is displayed currently. And finally the fourth part indicates whether the automated playback mode is running or not.

- **Simulator settings:** This dialog allows changing important simulator settings. Parameters influencing the outcome of the simulation run are on the one side the diffusion coefficients for species X and Y (these determine how fast these species will diffuse) and on the other side the “dimension” parameter. This parameter is used as a scaling parameter and allows scaling down the diffusion effect (in formula (1.9) the dimension-parameter can be found as “r”). The last parameter to change in the settings dialog is the step-size parameter. This parameter influence how large a step will be taken during each integration. While this parameter does not influence the pattern that gets created this parameter is crucial for the performance and stability

of the simulation-core. If set too low, it will take longer till patterns begin to arise. If set too high on the other side and an update of the concentrations will no longer be possible.

The visualization application will also save information about recent files, window position and dimension upon closing the program. This way it will appear on the same position upon startup as it was when the application was closed.

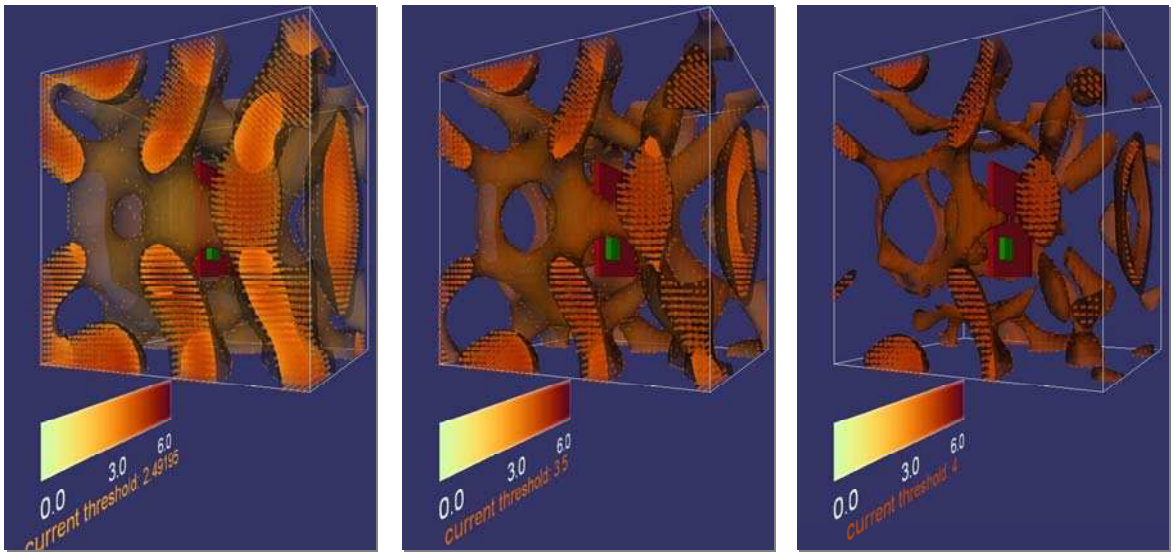
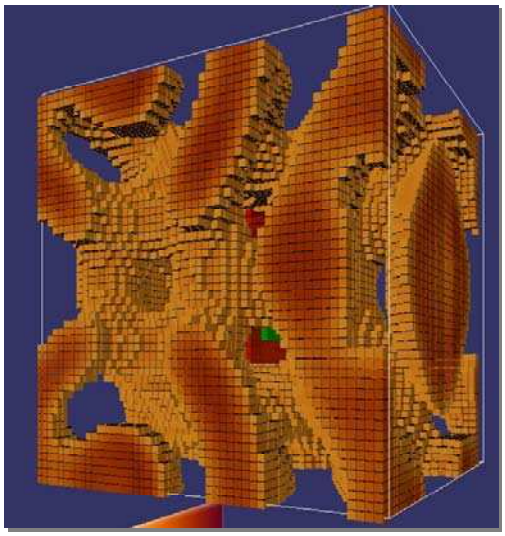
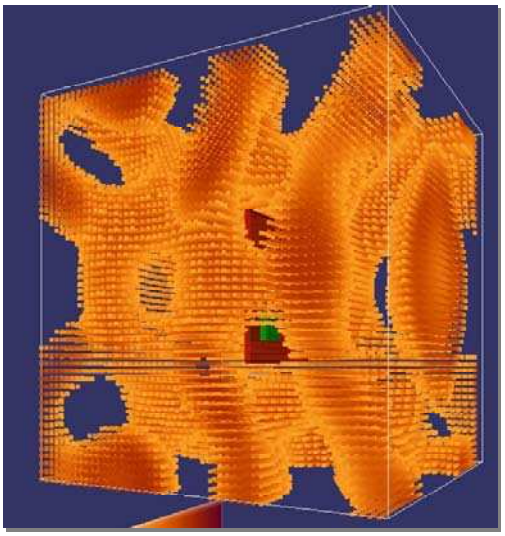


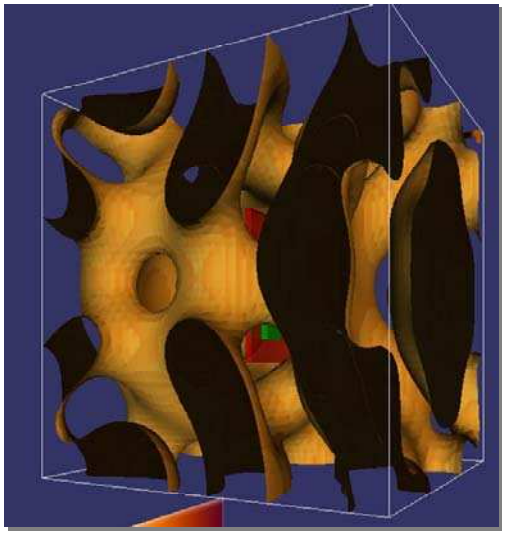
Figure 23: Using different thresholds (automatic(2.49), threshold=3.5, threshold=4.0)



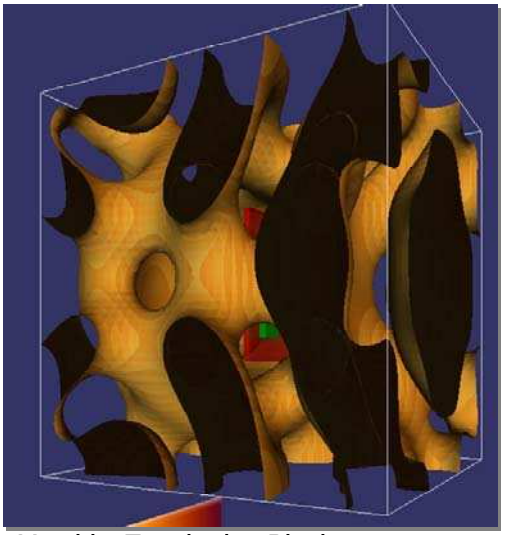
SimpleCubePlugin



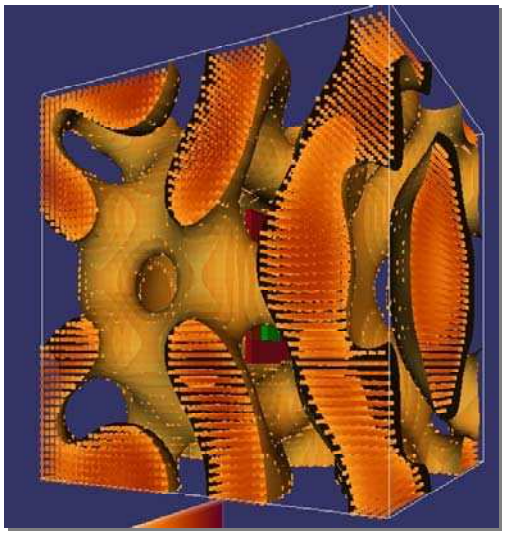
PointCloudPlugin



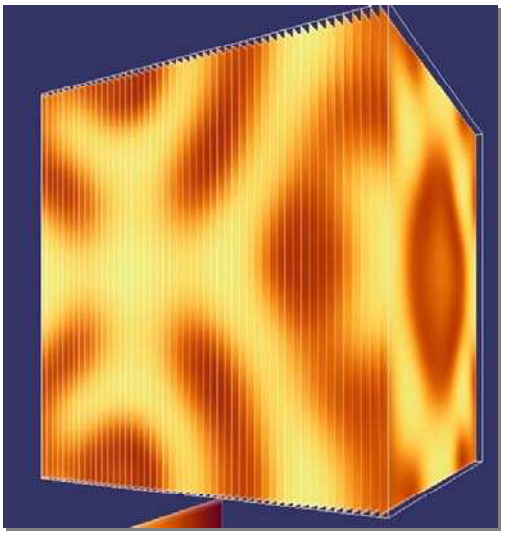
MarchingCubePlugin



MarchingTetrahedronPlugin



ComboPlugin



TexturePlugin

Figure 24: Available visualization plug-ins visualizing Example 3 of the appendix (transparency disabled)

5.5 Possible optimization for the simulation-core

This part will describe a possible optimization for the simulation-core. As previously detailed the core works in two phases. In the first phase all changes will be calculated. And then in a second phase these changes will be applied. The memory needed for this process is four times the size of the reaction-volume multiplied by the size of the data type stored. Thus it would be interesting to improve this. If the two phases could be combined into one phase that alone would noticeably improve the time needed for the simulation.

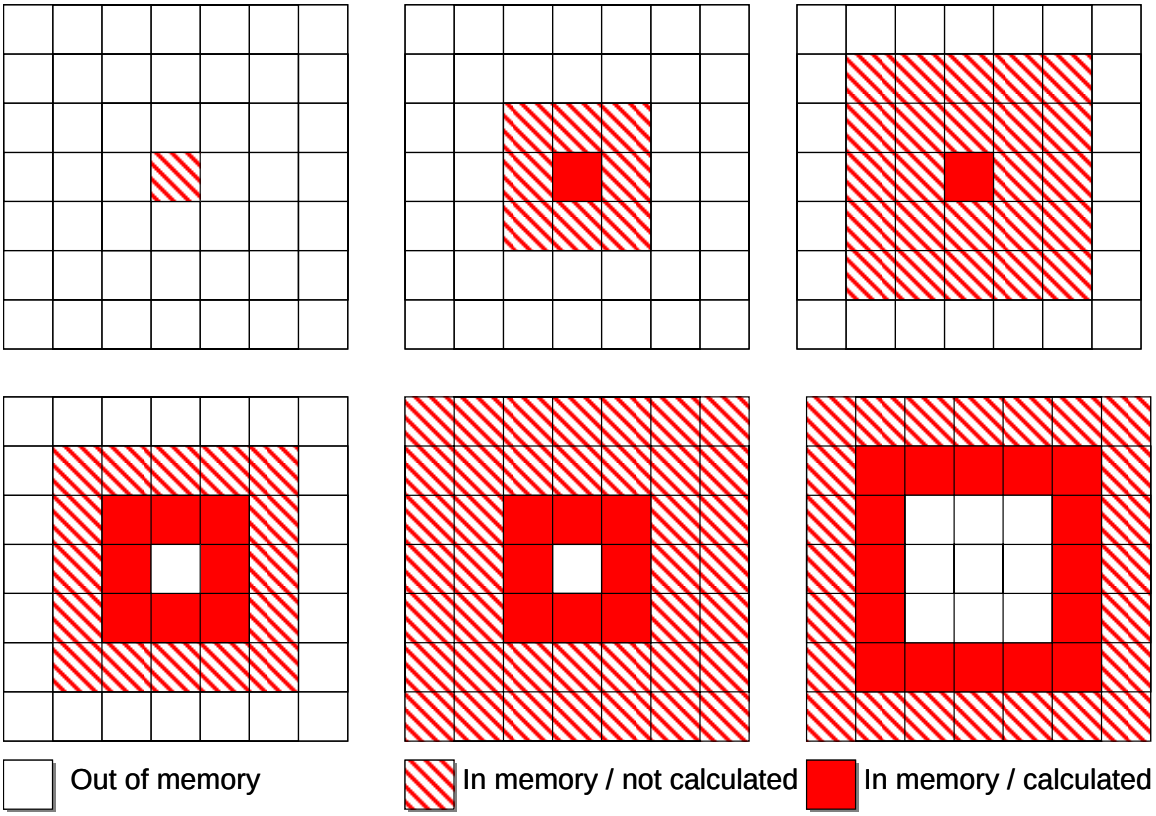


Figure 25: Update procedure of the optimized simulation-core

Figure 25 displays the idea for the two-dimensional case. The basic idea is to use an inside-out concept beginning at the center of the reaction-volume. (Actually since the structure of the reaction-volume is a hyper-torus the starting point does not really matter. This follows from the fact that all opposite edges are wrapped to each other.) From this start point out surrounding neighbors are red into memory (level 1). Now the center elements are computed and with them all elements of this level (level 0). The algorithm would now begin to read all outer neighbors of level 1 into the memory (level 2). Once this is completed all data necessary to compute the elements of level 1 has been red into the memory and the elements will be computed. This allows clearing all elements of level 0 from the memory. So the general structure is always: read level x, read level x+1, compute level x with data provided by level x-1, x, x+1 (at the starting point only levels x and x+1 are needed), clear level x-1, read level x+2, compute level x+1 …. This idea is followed until all elements have been computed.

How about the memory saved by this approach? About the memory needed for the concentrations of species X and Y nothing can be done. Thus we have two times the reaction-volume times the size of the data structure to begin with, but what about the rate of change? Here the memory needed is at most three times the outer hull of the reaction-volume. Together this would lead to the equation:

$$needed_{new} = 2\left(x \cdot y \cdot z + 3\left(x \cdot y + y \cdot z + x \cdot z\right)\right)$$

This equation is opposed to:

$$needed_{old} = 4 \cdot x \cdot y \cdot z$$

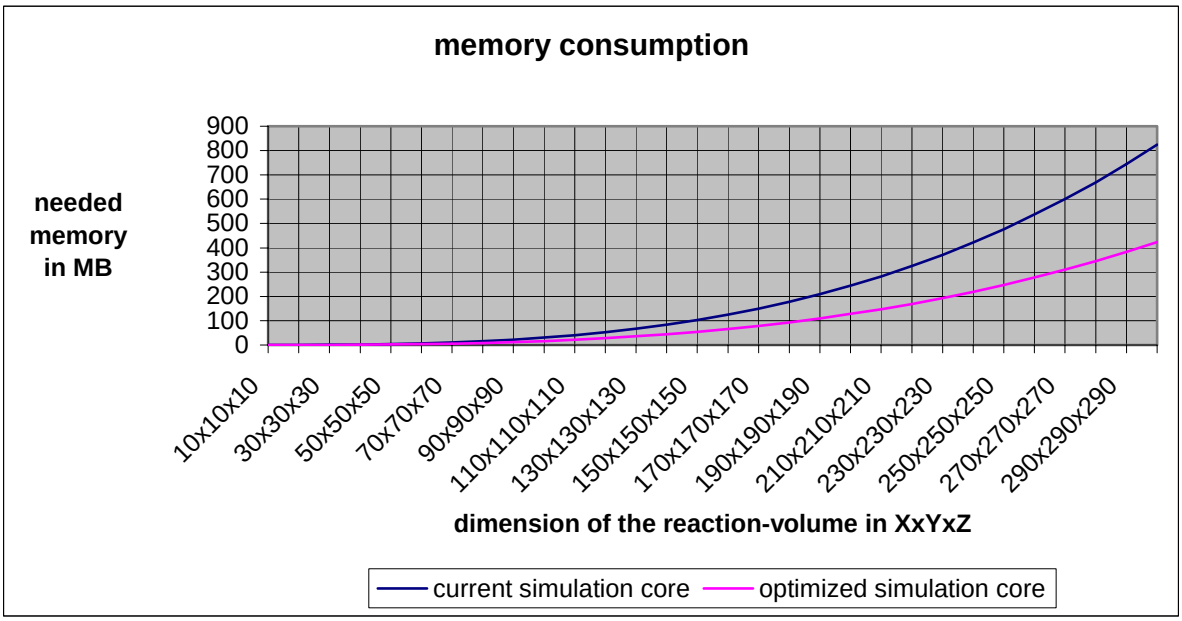


Figure 26: Comparison of current simulation-core vs. the optimized version

Figure 26 displays the memory requirements for both cases. It shows impressively that with this new method the amount of memory needed drops drastically. Since this new method would combine the steps of calculating a change and integrating it is expected that the runtime for this new algorithm is not much higher than for the old method.

5.6 Problems and Limitations

This section will detail encountered problems during implementation. Also limitations of the current implementation will be stated. Encountered were problems in the following areas: memory and runtime problems, problems due to the current implementation of the Systems Biology Workbench and problems due to the usage of specific features of Qt.

5.6.1 Complexities of three-dimensional simulation

As already hinted at in the description of the simulation-data structure (section 5.1.2) the demands placed by the three-dimensional simulation and visualization of reaction-diffusion systems on contemporary desktop-hardware are immense. Due to the cubic nature of the reaction-volume huge amounts of memory were needed. For the visualization even more than for the simulation since the objects needed for the simulation don't exceed four times eight byte per volume element, a value easily beaten by almost any visualization plug-in.

The approach taken to deal with these challenges are two-fold. On the one side addressing the memory for the simulation data had to be done “manually” by allocating the space using “malloc” and then addressing this memory according to the function:

$$position(X,Y,Z) = X \cdot \max Y \cdot \max Z + Y \cdot \max Y + Z$$

Although implemented using the “inline” feature this function will need some time for each access. Still, without this approach not even reaction-volume-dimensions up to 300x300x300 would be possible to simulate.

The second approach taken is to develop two possible visualization modes, the online mode for fast simulation and visualization of “smaller” reaction-volume dimensions and the offline mode for simulation of larger dimensions.

Even then the problem seems still not really tractable for large reaction-volumes. Mentioning that a 300x300x300 reaction-volume needs about 824MB of memory, along with the fact that most reaction-diffusion system take about 4000-7000 iterations until they display interesting patterns, should be sufficient to prove the point. For all practical reasons on contemporary desktop-hardware the largest reaction-volume has thus the dimension 200x200x200. See also the figure detailing the memory needs of the current simulation-core (Figure 26).

A third approach has also been evaluated. Here the idea was to compress the simulation-data using the ZLib compression algorithms. The drawback for this approach was that the (presumably) large double arrays had to be converted to character arrays first. The additional time and memory needed for the compression and decompression made this approach useless.

5.6.2 Using the Systems Biology Workbench

The concept of enhancing the capabilities of the simulator through the Systems Biology Workbench is certainly an interesting one. Especially the idea of visually designing the reactions had merit. From the beginning it was expected that the runtime would drop noticeably. The expected runtime drop was attributed to Jarnac having to interpret the equations build by JDesigner.

The actual time needed for sample runs defied these expectations. Even for very small reaction-volumes with a dimension of 50x50x1 needed about a second to be computed. To give SBW credit it was not designed to handle high throughput of data. Forthcoming versions promise possibilities for better ways of sharing memory across modules. Until then this plug-in should be used with care only.

5.6.3 Qt multithreading and the QProgressDialog

The basic idea was to use the visualization plug-ins in a separate thread in order to be prepared for an eventual failure. For example the visualization plug-in “`SimpleCubes`” this plug-in supported the display of each and every volume element above a certain threshold as cube along with transparency. For the case of larger reaction-volumes this would lead to enormous amounts of runtime and memory. Thus the generation of the graphical representation of the iterations was moved to a different thread. Furthermore the `QProgressDialog` was used. This dialog measured the progress and provided the user with a progress display and the possibility to cancel the operation. Moreover it also promised only to appear in case the execution took longer than a certain configurable amount of time. Upon implementation two things were noticeable: First the dialog appeared over and over again for small reaction-volumes, each time taking away the focus from the applica-

tion. Secondly canceling an ongoing visualization proved not to be a good idea, since the application tended to get more instable after a thread was cancelled.

Thus the multithreading feature is disabled for now until a better way can be found to cancel an ongoing visualization. For the time being warning dialogs have been implemented for visualization plug-ins that are known to lead to very long runtimes and excessive memory usage.

6 Evaluation

This section will first recapitulate the proceedings that have let to the achieved result (i.e. the application-suite for the configuration, simulation and visualization of simple three-dimensional reaction-diffusion systems). As a next step it will be argued that the result is indeed an outcome of the proceedings and finally that the proceedings follow the goals specified in the problem definition.

The goal to create a reaction-diffusion simulator working in three dimensions involved the following three main tasks:

- First of all the reaction-space had to be configured. Into this reaction-space elements (species, membranes and channels) had to be inserted.
- These settings had then in a second step to be processed by a simulator.
- And in a final step the output of the simulator had to be visualized.

Realizing this threefold-ness the author began looking at each of the three tasks individually. Never forgetting that the tasks are not completely isolated and always checking the interactions between these three main tasks (configuration, simulation and visualization). This separation can be found in every aspect of this thesis analysis, conception and implementation. Now, looking back, it turns out that this decision opened a way to solve the given problem. With the separation into the three parts, three applications were developed which are tied strongly together. Nonetheless with this approach it is still possible to easily replace one part of the chain by a different program. For example it would be thinkable to have configurations generated by a third party program that handles the insertion process more elegantly. Or a different simulator could be chosen. With one monolithic application this would not have been possible. One advantage of a monolithic application would probably be a stronger integration of the configuration into the visualization part of the program. In the developed set of programs it is possible to change a running simulation via selecting a different configuration file. If on the other side a monolithic application were built, insertion / moving / deleting of elements would be possible in one application. Another nice feature of an integrated application would be a preview feature that would allow the user to evaluate the configuration. Such a preview feature was not implemented because of the huge amount of time it would consume in most cases.

Of course each of the three applications still represents a prototype. Thus the next paragraph will take a closer look at each application to evaluate the prototype. While the graphical user interface makes the configuration unit rather accessible, it still is very hard to position elements exactly where the user would like them to be. Noticing that in early presentations the author added the possibility for directly entering all parameters. Still, better input metaphors should be developed in the future.

The developed prototype for the simulation unit works fine. The only negative point noticeable here is that the optimization method as described in section 5.5 was not yet implemented. This way only reaction-volumes up to a dimension of 300x300x300 can be simulated. Even then the amount of data created and written to the hard drive is immense. This would amount to up to 800 MB per iteration. These amounts of data make even that size intractable at the moment. For all practical reasons the dimensions of the reaction-volume should not exceed 200x200x200 elements. The idea of the reaction-plug-ins worked out nicely. The only major flaw here is the Jarna-

cReactionPlugin. Since Jarnac has to be asked for the rates of changes for each element a huge amount of communication is done. This is according to the current SBW communication message flow. SBW-modules are not allowed to “communicate” directly with each other. So each time a message is to be sent, it is sent first to the SBW-Broker and from there to the original recipient. Still the concept of visually creating reaction networks is an interesting one and was worth pursuing.

The final application to evaluate is the visualization unit. The two execution forms online- and offline visualization worked out as expected. With the possibility to change from one execution form to the other a high degree of flexibility was achieved. Of course there could be more visualization types but the plug-in system makes it possible to extend the available library easily. The only point that did not work out as planned was the multithreading idea. As already described in section 5.6.3 it was not possible to successfully separate the image generation from the main GUI thread. This results in possibly large delays and infrequently in a non-responding application. Until that problem is solved the only solution was to warn the user each time a visualization type is selected that is known to consume a large part of the available resources. Since a real interactive playback could not be achieved for larger reaction-volume dimensions the possibility for the saving of screenshots was provided. This together with the feature of automatically saving screenshots every x iterations helps to create movie files of these visualizations.

These three implemented applications together allow configuring, simulating and visualizing simple reaction-diffusion systems. They follow closely the steps defined in the conception phase. The conception phase on the other hand is strongly based on the analysis section. Together analysis, conception and implementation solve the problem stated at the beginning. It proved that the implementation achieved a working prototype. Hence the concept is validated through the implementation.

7 Summary and Outlook

This section concludes this thesis. In the two sections, first the work done will be summarized (recapitulating the individual steps analysis, conception and implementation) before a second section will give an outlook of how this project will be continued.

7.1 Summary

In order to research the formation of sub cellular structures and patterns or spatial effects in metabolic processes it is helpful to use simulations of three-dimensional reaction-diffusion systems.

The aim of this thesis was to develop algorithms and data structures for the configuration, simulation and visualization of simple three-dimensional reaction-diffusion systems. Towards this aim the thesis began to analyze input metaphors for three-dimensional modeling and the comparison of different file formats used in biological modeling. This provided the basis for the configuration of the reaction-space along with the setup of initial concentrations inside this reaction space. A next step analyzed what reaction-diffusion systems were analyzed in the past and were applicable to this case. Here it turned out that even half a century after its initial publishing Alan Turing's paper (c.f. [Tur52]) on the chemical basis of morphogenesis contained the most common used reaction-diffusion equations. In order to be able to support ever new forms of reaction-diffusion systems a plug-in system was conceived. This way the reaction-part of these equations could be substituted by a different one. The analysis of the visualization of simulation-data involved a comparison of different rendering APIs, a study of commonly used volume-visualization algorithms and an evaluation of volume-visualization libraries. This brought along the idea of applying the plug-in system for the visualization too. This provided an easy extendable selection of visualization types.

This analysis helped to determine a concept which was then implemented with the following result. On the basis of C++, extended by the Qt library for a highly capable windowing system and by OpenSceneGraph a new high level scene graph API, three applications have been developed. The first application, the configuration unit, allows setting up the reaction-volume along with desired features such as initial concentrations, membranes and channels. Furthermore the type of simulation to run will be specified using the configuration unit. This configuration will in a next step be saved onto hard drive using the file-format SBML level 2. Choosing that format provided the unique possibility to further extend the available reaction-diffusion systems by visually designing new equations using JDesigner, a program for visually defining reaction networks.

The second implemented application, the simulation unit, allows using the thus generated configurations to simulate the specified reaction-volume. Supported simulation modes implemented are diffusion, the Brusselator reaction-diffusion scheme and a simulation mode employing the simulation capabilities of Jarnac. Jarnac is used as SBW module that interprets the reaction-equations as specified in the visually created reaction network using JDesigner. Using this last simulation mode revealed a performance issue when using SBW for the transfer of large amounts of data.

The third application in this suite of applications is the visualization unit. In order to investigate the possibility of a three-dimensional visualization during course of the simulation two operation modes were implemented. On the one side the visualization application can be used to alternate between simulation of an iteration and its visualization. On the other side the visualization unit will

“just” display recorded simulation runs. During course of the visualization the user is allowed to change the type of visualization for a maximum amount of information. Furthermore the user has been provided with the possibility of analyzing a dataset by selecting a threshold level that will determine the iso-surfaces that will be visualized. And finally the user will have the possibility to define a clipping volume. This enables the user to “slice” the display in a way that will provide the demanded information.

The chapter implementation ends with stating a possible algorithm for optimizing the simulation-core. Although the author couldn’t find the time necessary to implement this algorithm in the allotted timeframe for this thesis, it is expected that this optimization will greatly reduce the space needed in memory as well as on the hard drive.

In an evaluation chapter was then detailed, that the result of this thesis, namely the developed program suite, does indeed follow the concept which was generated through an analysis of the requirements. Furthermore it showed that the concept was validated through the implementation, since the final program suite works correctly and solves the problems posed to it.

Summarizing the results of this thesis it can be said that although it is possible to simulate small to medium sized three-dimensional reaction-volumes and visualizing their results on current desktop-computers, the demand on the hardware is still one notch to high. Thus a lot of simplifying assumptions had to be made in order to achieve small results. Nonetheless the importance of research in this field is not to be neglected.

7.2 Outlook

This chapter will detail ideas for the improvement of the created application. The suggestions involve all levels of the application: configuration, simulation and visualization.

In research there is a still a high demand for the simulation and visualization of two-dimensional reaction-diffusion systems (perhaps due to the fact that these systems are tractable with currently available hardware). Although the current application supports the simulation and visualization of two-dimensional reaction-diffusion systems the underlying system will treat it still as a three-dimensional one (with only one element in the z-plane). A dedicated 2D mode would speed up these simulations and visualizations. Moreover it would be much easier to configure such a simulation run, since a dedicated 2D mode would allow “painting” species, membranes and channels on the surface.

There exist various ways to improve the simulation-core in order to reduce the demand for memory needed. In section 5.5 one optimization was presented which could not be implemented within the allotted time frame. Implementing this optimization would already half the memory needed for large reaction-diffusion simulations. Other optimizations are thinkable. For example it could be researched whether swapping mechanisms represent a feasible approach to this problem. Another possibility would be to research parallel calculation of different slices of the data-set and combining the results from these runs.

A large field for improvements would be to get rid of some of the assumptions made. Dynamic substrate changes or different diffusion rates through the reaction-volume would represent only two possibilities. The way membranes are currently implemented opens many possibilities for enhancements. The current model of basically blocking any molecules hitting the membrane and hindering the passage would be true only for very large molecules. Thus it would be interesting to allow for semi-permeable membranes. Also it should be possible to provide a different set of reac-

tion-equations for the membrane surfaces. Finally it would be interesting to have the possibility of “moving” membranes.

In his time at the Keck Graduate Institute the author ported some-core components of SBW from JAVA to C++ this enabled SBW to require less memory and simplified the installation procedures. Still the communication between the individual SBW modules is quite exhaustive. An improvement here along with the introduction of shared memory between SBW modules would greatly enhance its applicability to various fields. This would also allow the Jarnac reaction-plug-in to function with greater speed.

A final large field for improvement is represented by the visualization of the simulation-data. The prototypic visualization-plug-ins implemented so far should be enhanced further. Highest priority here would have the integration of an open visualization library such as vtk or OpenDX. This is another field that could not be touched in the allotted time-frame. Also applying multiple visualization plug-ins to one data-set in one display would be helpful. This is currently possible through actual implement a new visualization-plug-in that calls the visualization-plug-ins to combine. This is demonstrated by the “`ComboPlugin`” (see also Figure 24 for a visualization employing this plug-in.). This plug-in combined the marching cube representation and the representation through rendered points and thus allowed filling the “inside” of the surfaces generated by the marching cube algorithm with “life”. Having a visual way to combine these plug-ins would be helpful.

Appendix

A - Program CD

This section will describe the program CD accompanying this thesis. The CD contains the windows version for the three programs configuration unit, simulation unit and visualization unit along with the PDF version of this thesis, developer documentation and of course the source code for this project. As soon as the CD is placed into the CD-ROM drive the dialog displayed in Figure 27 should be opened. This dialog allows installing the created reaction-diffusion simulator and the Systems Biology Workbench. The visualization unit and the configuration unit are also accessible directly from the CD-ROM. This allows for a quick assessment of the created applications. To effectively demonstrate the different visualization-types it is only necessary to start the visualization unit. There choosing "continue simulation" and selecting the "`.info`" from one of the displayed directories will result in a first visualization. From there all features of the program are accessible (i.e. changing the visualization-type, clipping, navigating through the scene …).

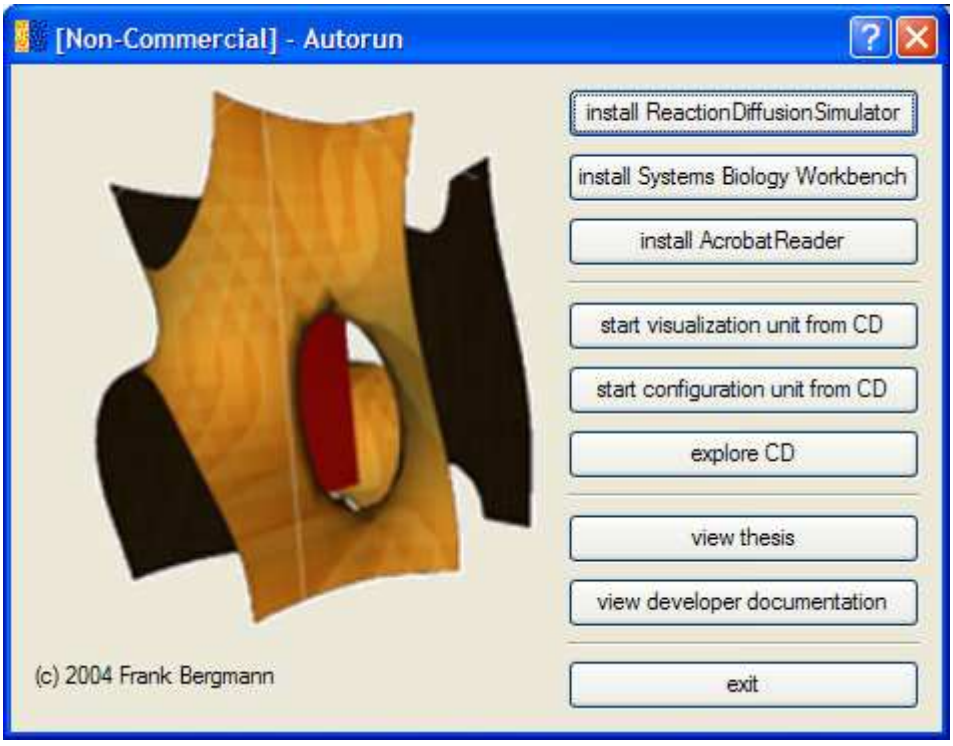


Figure 27: Autorun Dialog of the included CD-ROM

Along with the programs created the program CD also contains developer documentation created using doxygen. Furthermore a PDF version of the thesis will be available. Finally the program CD comes along with screenshots created throughout the developing process and some video files that show a various simulation runs.

B - Examples

This section will demonstrate how to create a configuration using the configuration unit and how to visualize these configurations applying the online mode of the visualization unit.

Configuration

Here a simple configuration will be created. The reaction-volume will have the dimensions 50x50x50. Furthermore a couple of elements will be inserted to demonstrate how membranes, channels and species work together. As simulation mode “diffusion only” will be selected. This will demonstrate effectively how the membranes influence the diffusion process of the species.

A second configuration to create will demonstrate the 2D capabilities of the developed suite of applications. Therefore the reaction-volume will have the dimensions 100x100x1. Here only some molecules will be inserted. As simulation mode the Brusselator reaction-scheme will be selected. This will allow demonstrating the patterns that will be generated using the reaction-diffusion simulator.

In order to create a configuration the following two steps are necessary. First a new reaction-volume has to be created. Therefore “File\New” is selected form the File-menu of the “configuration unit”. In the now appearing dialog the dimensions are to be entered and a simulation mode to be chosen. The second step is to insert the elements that should be in the scene. Therefore the individual insertion tools are to be selected from the Insert-menu. Once the insert mode is selected two possibilities for the insertion exist. Either the user enters the data (position and concentration for species, position and dimension for membranes and channels) in the now visible property pane, or the user uses the mouse to position the element and commits the position by pressing space.

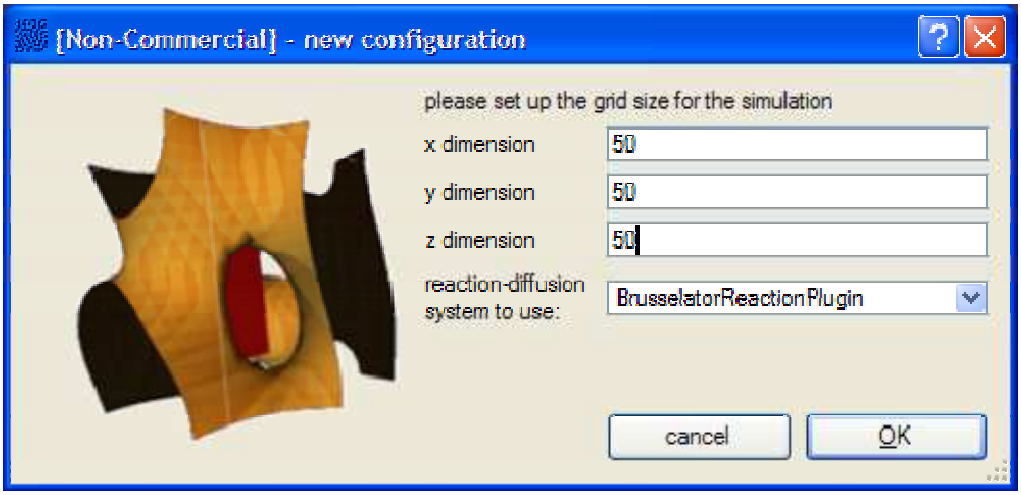


Figure 28: The file new dialog of the configuration unit

Figure 28 displays the “File\New” dialog. For the first example the dimensions should be set to 50x50x50 as displayed above. Under “reaction-diffusion system to use:” the “`ZeroReaction-Plugin`” should be selected. “`ZeroReactionPlugin`” hints at the fact that the reactions in the Turing system (c.f. section 4.3.1.1) are disabled (set to zero). This leads to a simulation of diffusion. After committing this selection with *OK* the reaction-volume will be displayed as wire-frame-cube.

The next task is to insert some elements into the reaction-volume. In order to demonstrate the diffusion plug-in one membrane will be inserted near the center of the reaction-volume. In the middle of the reaction-volume a channel will be placed. Finally several elements of species X will be inserted.

- To insert the membrane, the membrane insertion tool has to be selected. It is to be found under "Insert\Membrane" or by pressing the key "1". To actually insert the membrane the position 25,25,25 and dimension 20x20x1 will be entered in the property pane that appeared after activating the insert mode. After committing by pressing "insert" on the property pane the membrane will be inserted.
- The next step is the insertion of the channel. The channel insertion tool will be activated by selecting "Insert\Channel" or pressing "2" on the keyboard. In the changed property pane the position will be changed to 20,20,25 and the dimension to 4x4x3. Again these changes will be committed by clicking "insert".
- Finally some molecules are to be entered. This is done by selecting "Insert\Molecules of Species X" or pressing "x" on the keyboard. Using the changed property pane the following four molecules will be inserted:
    - position: 25,25,15, concentration: 6.0
    - position: 26,25,15, concentration: 6.0
    - position: 25,26,15, concentration: 6.0
    - position: 26,26,15, concentration: 6.0

With these elements a first simple configuration is complete. What is left is to save the configuration. Therefore "File\Save" is chosen and the configuration is saved under: "`50x50x50_Diffusion.xml`". Figure 29 displays the configuration of this first example.

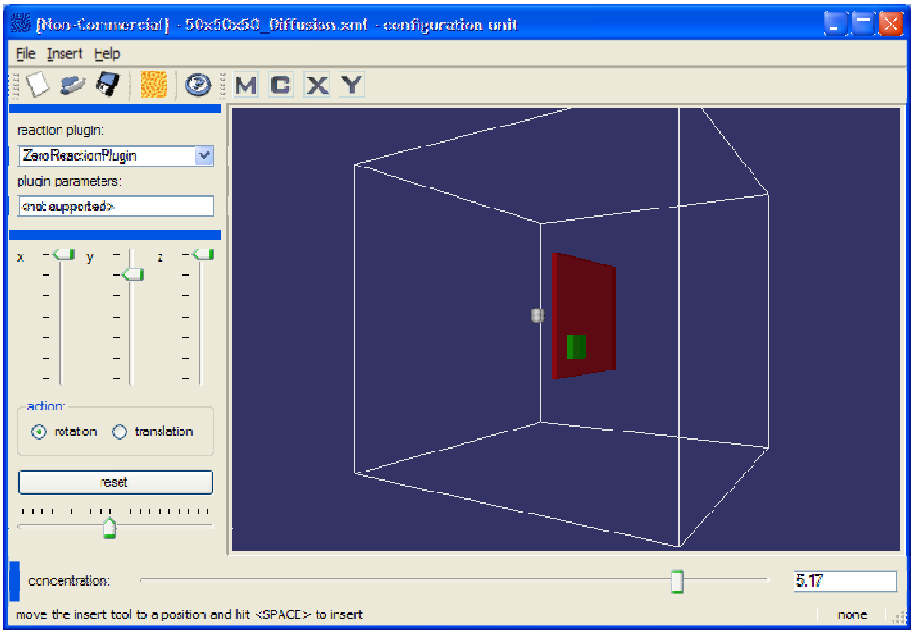


Figure 29: The configuration created for the first example.

A second example is created in the same way. In order to create a two-dimensional reaction space the dimension of the reaction-volume will be given as 100x100x1 in Figure 28. The simulation mode will be set to "`BrusselatorReactionPlugin`". Only four elements will be needed for

this example. Elements of species X will be inserted in the center of the reaction-volume on the positions 50,50,1; 51,50,1; 50,51,1 and 51,51,1. Each element is inserted with a concentration of 6.0. This configuration will be saved as "`100x100x1-Brusselator.xml`".

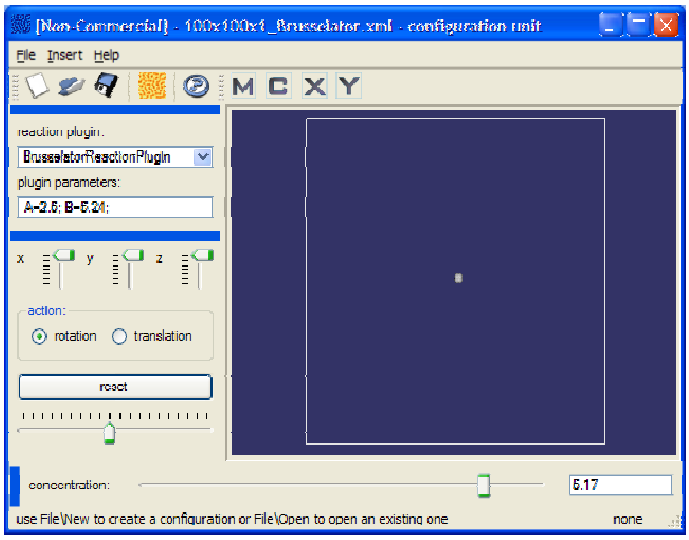


Figure 30: The configuration of the second example (2D Brusselator)

Visualization

This section will demonstrate how to visualize the configurations created in the last section. Since the first configuration was created with the thought in mind to effectively present how membranes influence the spreading out of the inserted species, an iso-surface representation of the dataset will prove the point easiest. Thus the marching cube visualization will be selected.

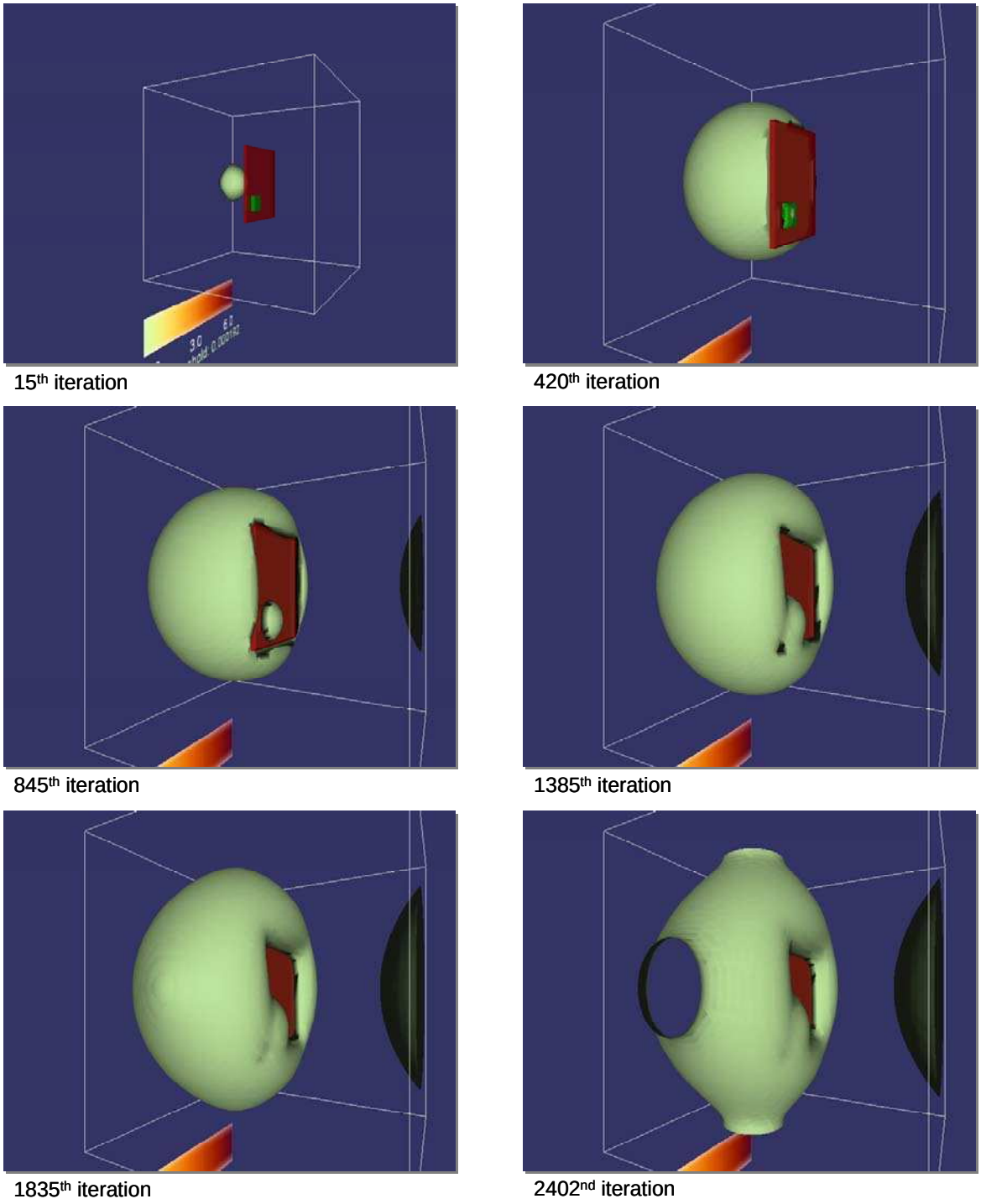
The second created configuration was a two-dimensional one. Thus the best plug-in to demonstrate this mode will be the "texture visualization plug-in". This example also demonstrates the feature to set different visualization step sizes. That allows skipping several thousand image-generations and will display the expected patterns soon.

The last simulation run here is of an altered configuration of the first example. The only changes made here was to select a different simulation mode, namely the Brusselator simulation mode. This demonstrates the three-dimensional capabilities visualizing reaction-diffusion schemes.

In order to perform an online visualization of the created examples, the first step is to start the application "visualization unit". The next step is the selection of "File\Run Simulation" and the opening of the created configurations, "`50x50x50_Diffusion.xml`", "`100x100x1_Brusselator.xml`" and "`50x50x50_Brusselator.xml`", in the appearing file-open-dialog. After selecting the configuration to run, an initial iteration will be computed. And displayed with a default-visualization plug-in (the marching cube visualization) and enabled transparencies. Transparencies are here weighted according to an elements concentration ranging from 0.0 to 6.0.

In order to visualize the first configuration ("50x50x50_Diffusion.xml") transparencies were disabled. The reason is the selected simulation mode. Since only diffusion is simulated the concentrations throughout the reaction-volume will be rather low, and through the weighted transparencies not visible. Transparencies are deactivated by pressing "t" or deactivating "Settings\enable transparency".

Figure 31 shows some iterations of the simulation run, of the first example. These screenshots were made using the feature to save a screenshots each time a new iteration is displayed. This feature is enabled through pressing "A" or selecting "Settings\enable screenshot generation". The pictures show the expanding of the concentration through the reaction-volume. When this "wave" "hits" the membrane, it cannot go through. It expands along the surface of the membrane until it reaches the end of the membrane, or the channel. Then it continues its spreading out through the reaction-volume.



15th iteration

420th iteration

845th iteration

1385th iteration

1835th iteration

2402nd iteration

Figure 31: Marching cube visualization of the configuration created in the first example (50x50x50_Diffusion.xml)

Figure 32 displays the visualization of example two (“`100x100x1_Brusselator.xml`”). Here the texture visualization was chosen. In first 5700 iterations one can see that the concentrations through the reaction-volume oscillate from low to high concentrations. Then slowly patterns begin to form. These pictures demonstrate the chosen nature of the reaction-volume. First it is noticeable that concentric patterns begin to form until the “rings” reach the boundaries of the reaction-volume. Since the edges are wrapped around each other (i.e. the reaction-volume represents a torus) the concentration-“waves” influence each other as they enter the reaction-volume from the other side.

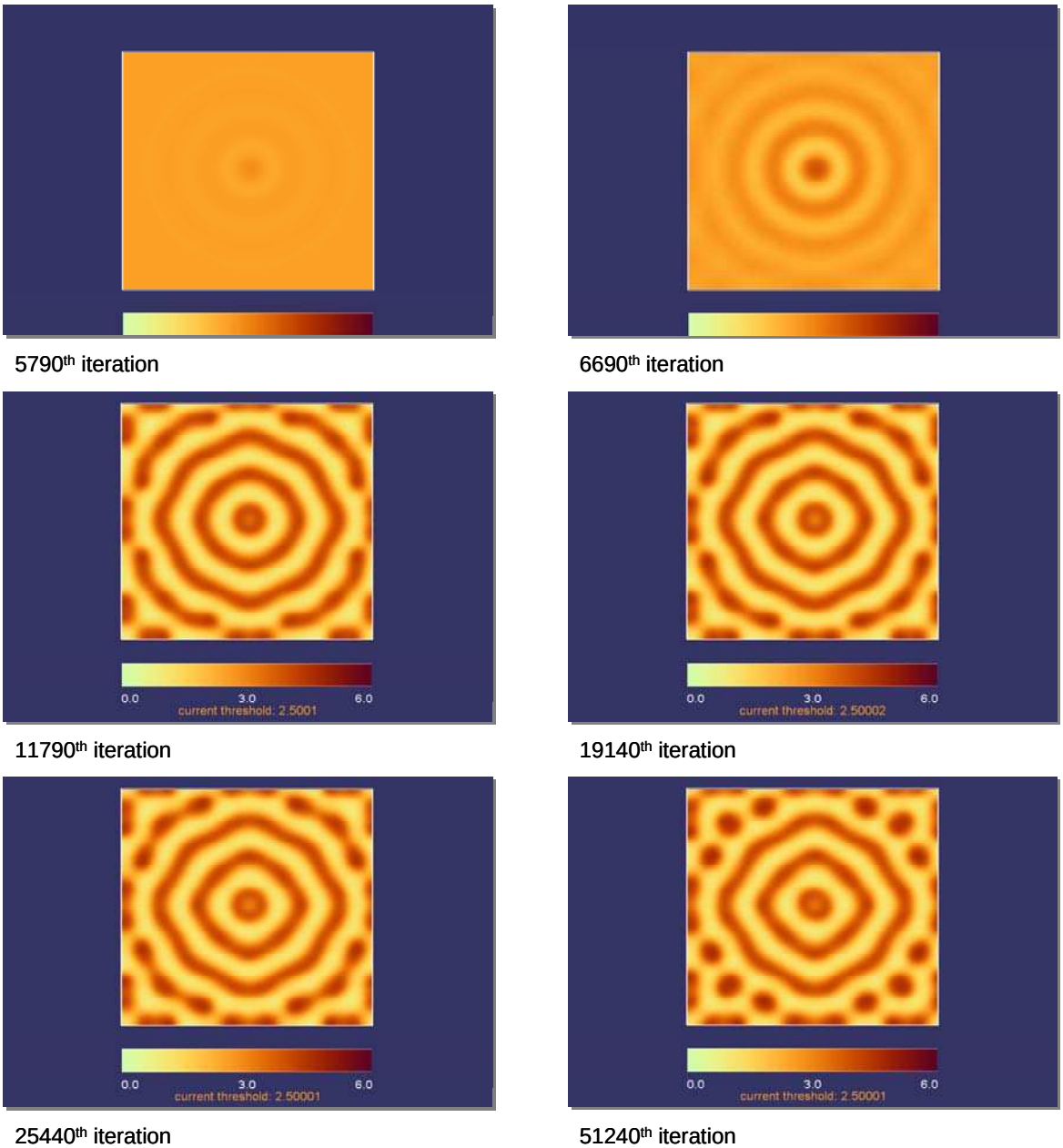
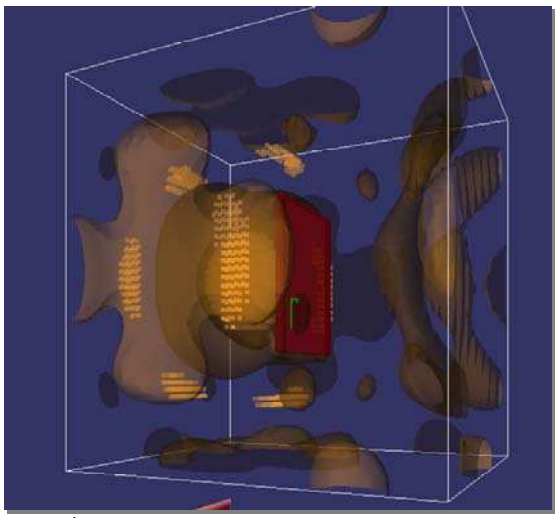


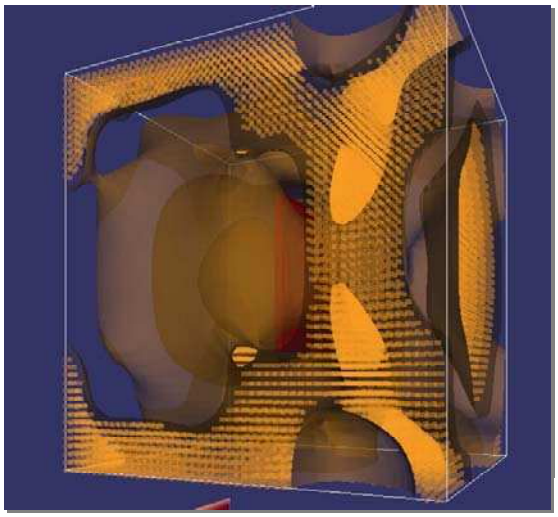
Figure 32: Texture visualization of example two (100x100x1_Brusselator.xml)

Finally Figure 33 displays the visualization of a changed example one. As already mentioned here the simulation mode was changed from “`ZeroReactionPlugin`” to “`BrusselatorReactionPlugin`”. As visualization plug-in the ”`ComboPlugin`” was chosen. This visualization plug-in uses the marching cube visualization to create an iso-surface according to the desired threshold. The inside of the thus created object is filled with points, representing the concentrations

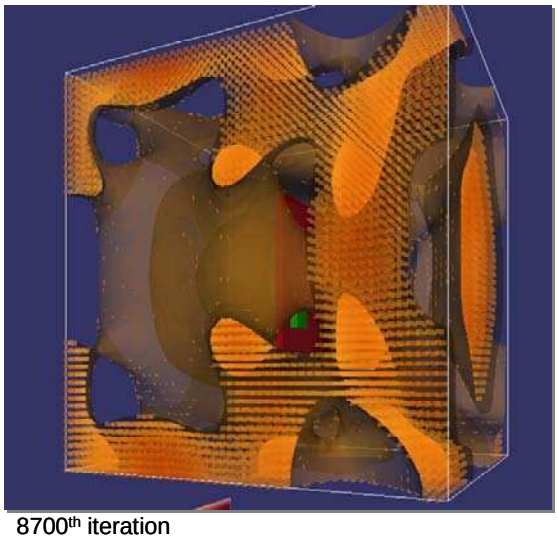
at the data-points. Again it takes about 6000 iterations until the first patterns begin to arise. This series of pictures also display how transparency is used to highlight higher concentrations.



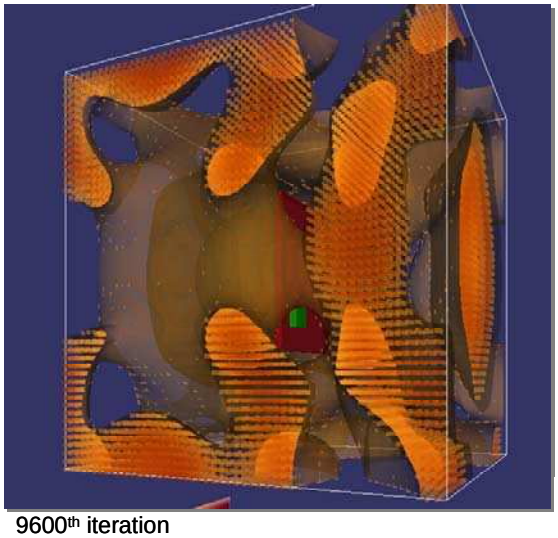
6300th iteration



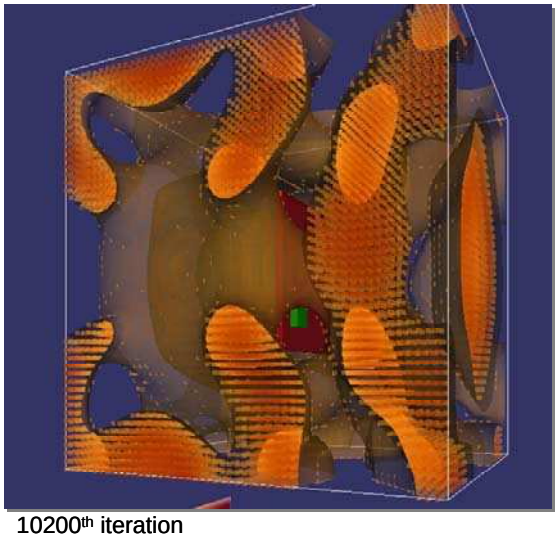
7500th iteration



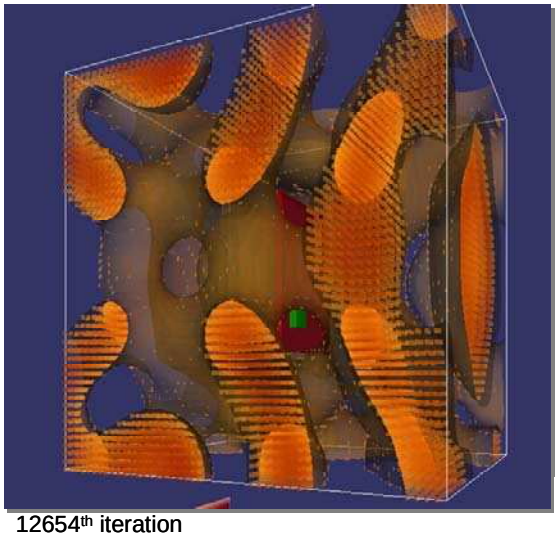
8700th iteration



9600th iteration



10200th iteration



12654th iteration

Figure 33: Visualization of example three through a combination of the marching cube visualization and the point-cloud visualization (50x50x50_Brusselator.xml)

C - Schema Definition of SBML Annotations

Here the schema definitions for the annotations used in order to be able to save the configuration data in an SBML document. Three elements were used:

- `rdb:dimensions`
- `rdb:reaction-plug-in` and
- `rdb:element`

With these definitions all necessary data could be saved.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- schema definition of sbml annotations used in order
     to save configuration data generated as part of the
     reaction-diffusion simulator -->
<xs:schema targetNamespace="http://www.sys-bio.org/sbml"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns:rdb="http://www.sys-bio.org/sbml"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">

      <xs:element name="dimensions">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="x" type="xs:integer" />
                        <xs:element name="y" type="xs:integer" />
                        <xs:element name="z" type="xs:integer" />
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="reaction-plugin">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="name" type="xs:string" />
                        <xs:element name="config" type="xs:string" />
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="element">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="x" type="xs:integer" />
                        <xs:element name="y" type="xs:integer" />
                        <xs:element name="z" type="xs:integer" />
                        <xs:element name="concentration" type="xs:double"
                                    minOccurs="0" />
                        <xs:element name="width" type="xs:integer"
                                    minOccurs="0" />
                        <xs:element name="height" type="xs:integer"
                                    minOccurs="0" />
                        <xs:element name="depth" type="xs:integer"
                                    minOccurs="0" />
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

D - Bibliography

[Ahr00] **Ahrens, J.; Law, C.; Schroeder, W.; Martin, K.; Papka, M.: "A parallel approach for efficiently visualizing extremely large time-varying datasets"**
Los Alamos National Laboratory, Los Alamos, Technical Report LAUR-00-1620, 2000

[AMI04] **Official AMIRE website**
http://www.amire.net, last accessed: 17.09.2004

[AVS04] **Advanced Visual Systems: "AVS/Express Multipipe Windows PC Cluster Edition"**
http://www.avs.com/, last accessed 06.09.2004

[Bla04] **Blanchette, Jasmin; Summerfield, Mark: "C++ GUI Programming with Qt3"**
Pearson Education Inc., New Jersey, 2004

[Bor03] **Bornstein, Ben: "libsbml Developer's Manual", August, 2003**
http://sbml.org/ , last accessed: 17.09.2004

[Bor97] **Bourke, Paul: "Polygonising a scalar field", 1997**
http://astronomy.swin.edu.au/~pbourke/modelling/polygonise/, last accessed: 17.09.2004

[Bur04] **Burns, Don; Osfield, Robert: "Introduction to the OpenSceneGraph"**
http://www.openscenegraph.org, last accessed: 17.09.2004

[Car03] **Carr, Hamish; Theußl, Thomas; Möller, Thorsten: "Isosurfaces on Optimal Regular Samples"**
Proceedings of the symposium on Data visualisation 2003 Grenoble, France, 39-48, 2003

[Chi97] **Chiueh, Tzi-cker, Yang, Chuan-kai; He, Taosong; Pfister, Hanspeter; Kaufman, Arie: "Integrated Volume Compression and Visualization"**
Proceedings of the 8th IEEE Visualization'97 Conference, October 1997

[Cue03] **Cuellar, A.; Nielsen, P.; Halstead, M.; Bullivant, D., Nickerson, D.; Hedley, W.; Nelson, M.; Lloyd, C.: "CellML Specification", Sept. 30th, 2003**
http://www.CellML.org/public/specification/20030930/CellML_specification.html, last accessed: 17.09.2004

[Ebe04] **Ebert: "Dividing Cubes Algorithm"**
http://www.cs.umbs.edu/~ebert/693/THu/dividingcubes.html , last accessed: 27.03.2004

[Enc97] **Encarnação, José; Straßer, Wolfgang; Klein, Reinhard.: "Graphische Datenverarbeitung 2"**
Oldenburg, Munich; Vienna, 1997

[Eng00] **Engel, K.; Hastreiter, P.; Tomandl, B.; Eberhardt, K.; Ertl, T.: "Combining Local and Remote Visualization Techniques for Interactive Volume Rendering in Medical Applications"**
Proceedings of the conference on Visualization '00, Salt Lake City, Utah, United States, 449-452, 2000

[Elv92] **Elvins, Todd T: "A survey of Algorithms for Volume Visualization"**
Computer Graphics, vol. 26 no. 3, 194-201, August 1992

[Fin04] **Finkelstein, A.; Hetherington, J.; Li, L.; Margoninski, O.; Saffrey, P.; Seymour, P.; Warner, A.: "Computational Challenges of Systems Biology"**
Computer, vol. 37, no. 5, 26-33, May 2004

[Fin03a] **Finney, Andrew; Hucka, Michael: "Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions"**
http://www.sbml.org , 2003, last accessed: 17.09.2004

[Fin03b] **Finney, Andrew; Hucka, Michael; Sauro, Herbert; Bolouri, Hamid: "Systems Biology Workbench C++ Programmer's Manual"**
http://sbw.sourceforge.net , 18.02.2003, last accessed: 17.09.2004

[Fol90] **Foley, J.; van Dam, A.; Feiner, S.; Hughes, J.: "Computer Graphics – Principles and Practice"**
Addison Wesley, 1990

[Frü96] **Frühauf, Thomas: "Raycasting Vector Fields"**
Proceedings of the 7th conference on Visualization '96, San Francisco, California, United States, 115-ff, 1996

[Gut01] **Guthe, Stefan; Straßer: "Real-time Decompression and Visualization of Animated Volume Data"**
Proceedings of the conference on Visualization '01
San Diego, California, Session P12: approximation and compression, 349-356, 2001

[Har93] **Harrison, Lionel G.: "Kinetic Theory of Living Pattern"**
Cambridge University Press, 1993

[Her79] **Herman, G.; Liu, H.: "Three dimensional display of human organs from computed tomograms."**
Computer Graphics and Image Processing, vol. 9 no. 1, 1-21, 1979

[Huc03] **Hucka, M.; Finney, A.; Sauro, H.: "The Systems Biology Markup Language (SBML): a medium for representation and exchange of biochemical network models."**
Bioinformatics vol. 19 no. 4, 524-531, 2003

[Kan96] **Kaneda, Kazufumi; Dobashi, Yoshinori; Yamamoto, Kazunori, Yamashita, Hideo: “Fast Volume Rendering with Adjustable Color Maps”**
Proceedings of the 1996 symposium on Volume visualization, San Francisco, California, United States, 7-ff., 1996

[Kit04] **Kiteware Inc.: “The Visualization Toolkit”**
http://www.vtk.org/what-is-vtk.php, last accessed 07.12.2004

[Lep03] **Leppänen, T.; Karttunen, M.; Barrio, R.A.; Kaski, K.: “Spatio-temporal dynamics in a Turing model”**
www.lce.hut.fi/research/polymer/turing_paper6.pdf, last accessed: 17.09.2004

[Lor87] **Lorensen, William E.; Cline, Harvey E.: “Marching cubes: a high resolution 3D surface construction algorithm”**
International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 163-169, 1987,

[Mei01] **Meinhardt, Hans: "Biological Pattern Formation"**
http://www.biologie.uni-hamburg.de/b-online/e28_1/pattern.htm, 2001, last accessed: 17.09.2004

[Mei82] **Meinhardt, Hans: "Models of Biological Pattern Formation"**
Academic Press, London, 1982

[Mic02] **Microsoft Corporation: "DirectX 9.0 Programmer’s Reference", 2002**
Microsoft Corporation, 2002

[Mon02] **Monster, Marco: "Achieving a Stable Simulator", 2002**
http://home.planet.nl/~monstrous, last accessed: 17.09.2004

[Mon92] **Montani, C.; Perego, R.; Scopigno, R.: "Parallel Volume Visualization on a Hypercube Architecture"**
Workshop on Volume Visualization, Boston, 10/1992

[Mur01] **Murray, J.D.: "Mathematical Biology"**
Springer-Verlag, Berlin Heidelberg, 2001

[Nei97] **Neider, Jackie; Davis Tom: "OpenGL Programming Guide (redbook)"**
Addison-Wesley Publishing Company, 1997

[Nic77] **Nicolis, G.; Prigogine, I.: "self-organization in nonequilibrium systems”**
Wiley, New York, 1977

[Nie04] **Nielsen, P.; Halstead, M.; Cuellar, A.; Dunstan, M.; Bullicant, D.; Hunter, P.: "CellML Evolution”**
http://www.sbml.org/workshop/seven/poul-cellml-evolution.pdf, last accessed: 17.09.2004

[Owe99] **Owen, Scott: "Volume Visualization and Rendering"**
http://www.siggraph.org/education/materials/HyperVis/vistech/volume/volume.htm, last accessed: 17.09.2004

[Pac02] **Programmierpraktikum Graphische Datenverarbeitung: "Pacman 3D"**
Programmierpraktikum Graphische Datenverarbeitung, University Frankfurt, WS 2001/2002, http://www.stormzone.de/uni/pacman3d/index.html, last accessed 18.09.2004

[Pek01] **Pekar, Vladimir; Wiemger, Rafael; Hempel, Daniel: "Fast Detection of Meaningful Isosurfaces for Volume Data Visualization"**
Proceedings of the 12th IEEE Visualization 2001 Conference (VIS 2001), October 2001

[Pre99] **Press, William H.; Teukolsky, Saul A.; Vallerling, William T.; Flannery, Brian P.: "Numerical Recipes in C"**
Cambridge University Press, 1999

[Qt04] **Trolltech: "Qt"**
http://www.trolltech.com/qt, last accessed 20.04.2004

[Sak02] **Sakurai, Tatsunari; Mihaliuk, Eugene; Chirila, Florin; Showalter, Kenneth: "Design and Control of Wave Propagation Patterns in Excitable Media"**
SCIENCE, vol 296, 2009-2012, 14.06.2002

[Sau00] **Sauro, Herbert M.: "Jarnac: a system for interactive metabolic analysis"**
Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics. J.-H.S, Hofmeyr, J.M. Rohwer, and J.L. Snoep, eds. (Stellenbosch University Press), 221-228, 2000

[Sau03a] **Sauro, Herbert M.: "JDesigner SBML Annotation"**
www.cds.caltech.edu/~hsauro/JDSBMLEx.pdf, January 8, 2003

[Sau03b] **Sauro, Herbert M.; Hucka, Michael; Finney, Andrew; Wellock, Cameron; Bolouri, Hamid; Doyle, John and Kitano, Hiroaki: "Next Generation Simulation Tools: The Systems Biology Workbench and BioSPICE Integration"**
Omics A Journal of Integrative Biology, vol. 7, no. 4, 355-372, 1.12. 2003

[Sch96] **Schroeder, William; Martin, Kenneth; Lorensen, William: "Design and Implementation of an Object-Oriented Toolkit for 3D graphics and visualization"**
Proceedings of the 7th conference on Visualization '96, San Francisco, California, United States, 93-ff., 1996

[Sch84] **Schuster, Heinz G.: "Deterministic chaos: an introduction"**
Physik-Verlag GmbH, Weinheim, 1984

[Seg04] **Segal, Mark; Akeley, Kurt: "The OpenGL Graphics System: A specification (Version 2.0 – September 7, 2004)"**
http://www.opengl.org/, last accessed 17.09.2004

[Tho86] **Thompson, J.M.T.; Steward, H.B.: "Nonlinear dynamics and chaos"**
John Wiley & Sons Ltd, Great Britain, 1986

[Tur52] **Turing, A.M.: "The Chemical Basis of Morphogenesis"**
Philosophical Transactions of the Royal Society of London, Series B, Biological Sciences, vol. 237, no. 641, 37-52, 14.08.1952

[Tur91] **Turk, Greg: "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion"**
Computer Graphics, vol. 25, no. 4, 289-298, July 1991

[Tur92] **Turk, Greg: "Texturing Surfaces Using Reaction-Diffusion"**
University of North Carolina, Chapel Hill, 1992

[Ups88] **Upson, C.; Keeler, M.: "V-BUFFER: visible volume rendering"**
Computer Graphics, Vol 22, No. 4, 59-64, August 1988

[Van02] **van der Vlist, Eric: "XML Schema"**
O'Reilly & Associates, Sebastopol, 2002

[Van04] **van der Zijp, Jeron: "FOX Windowing Toolkit"**
http://www.fox-toolkit.org, last accessed 20.04.2004

[Wri96] **Wright, Richard S., Jr; Sweet, Michael: "OpenGL superbible"**
Waite Group Press, 1996

[Zho97] **Zhou, Yong; Chen, Baoquan; Kaufman, Arie: "Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data"**
Proceedings of the 8th conference on Visualization '97, Phoenix, Arizona, United States, 135-ff., 1997

E - List of Figures

F – Index