D I S S E R T A T I O N

Numerical Methods for Topography Simulation

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
von

Otmar Ertl

Reinprechtsdorfer Straße 20/16
A-1050 Wien, Österreich

Matrikelnummer 0026810
geboren am 22. Februar 1982 in Vöcklabruck

Wien, im Mai 2010 _____

Kurzfassung

Die Simulation von Herstellungsschritten in der Halbleiterfertigung erlaubt nicht nur ein besseres Verständnis der zugrunde liegenden Physik, sondern auch die Prozessoptimierung ohne teure Experimente. Viele Schritte in der Prozesskette, um integrierte Schaltungen herzustellen, verändern die Topographie der Waferoberfläche durch Ätzen oder Abscheidung neuer Materialschichten. Diese Arbeit präsentiert neue numerische Verfahren, die eine effiziente und genaue Simulation solcher Prozesse erlauben.

Ein Hauptproblem der Topographiesimulation stellt die korrekte zeitliche Beschreibung der geometrischen Veränderungen, insbesondere in drei Dimensionen, dar. Daher wurde basierend auf der Level-Set-Methode eine Technik entwickelt, die neueste Algorithmen und Datenstrukturen, wie die Sparse-Field-Methode oder hierarchische Lauflängenkodierung, benutzt, um Rechenzeit sowie Speicherverbrauch zu minimieren und die Zeitentwicklung großer dreidimensionaler Geometrien zu ermöglichen. Eine genaue Berücksichtigung verschiedenener Materialschichten und der davon abhängigen Oberflächenraten ist für Ätzprozesse besonders wichtig und konnte mit einer neuartigen Multi-Level-Set-Methode erreicht werden. Zudem wurden Algorithmen für diverse geometrische Operationen und für den Test auf gerichtete Sichtbarkeit und Konnektivität verwirklicht. Um moderne Multikernprozessoren auszunützen, wird ein neuer Ansatz für die Parallelisierung der hierarchischen Lauflängenkodierung präsentiert.

Ein weiteres Problem ist die Bestimmung der Oberflächenraten für realistische Topographiesimulationen. Erweiterte Modelle für die Beschreibung der Oberflächenkinetik erfordern die Kenntnis der Teilchenflussverteilung an der Oberfläche. Obwohl meistens ballistischer Teilchentransport angenommen werden kann, ist die Berechnung dennoch sehr aufwändig, wenn Reemissionen oder spiegelartige Reflexionen berücksichtigt werden sollen. Simulationen sind daher oft limitiert auf vereinfachte Modelle oder kleine Strukturen. Ein vielversprechender Ansatz, um diese Einschränkung zu überwinden, ist eine Monte-Carlo-Methode, die durch die Simulation vieler Teilchentrajektorien die Flussverteilung bestimmt. Diese Arbeit präsentiert ein neues Verfahren, das diese Monte-Carlo-Berechnung direkt auf die implizite Oberflächendarstellung der Level-Set-Methode anwendet. Zudem wurde der Rechenaufwand durch die Anwendung von erweiterten Raytracing-Algorithmen und -Datenstrukturen, die für die Bedürfnisse von Topographiesimulationen adaptiert und optimiert wurden, reduziert.

Schlussendlich werden die gezeigten numerischen Methoden anhand verschiedener in der Literatur beschriebene Prozessmodelle getestet, um die vielfältige Anwendbarkeit, insbesondere für große dreidimensionale Strukturen, zu demonstrieren.

Abstract

The simulation of semiconductor manufacturing steps allows for a better understanding of the underlying physics as well as for process optimizations, without the need for costly experiments. Many steps in the process chain for building integrated circuits alter the topography of the wafer surface by etching or deposition of new material layers. In this work, new numerical techniques for an efficient and more accurate simulation of such processes are presented.

A major concern of topography simulations is an accurate description of the geometric changes over time, especially in three dimensions. For this purpose, a fast framework based on the level set method was developed. Using the latest algorithms and data structures, such as the sparse field method and hierarchical run-length encoding, the computation time and the memory consumption was minimized, which enabled the handling of the time evolution of large three-dimensional geometries. An accurate description of different material regions and material-dependent surface rates, which is especially important for the simulation of etching processes, was achieved using a novel multi-level-set technique. Moreover, algorithms for geometrical operations and testing of directional visibility and connectivity have been realized. In order to capitalize on modern multi-core processors, a new approach for the parallelization of the hierarchical run-length encoding is presented.

Another concern of realistic topography simulation is the determination of surface rates. Advanced surface kinetics models require the calculation of the particle flux distributions on the surface. Although the particle transport can be approximated to be ballistic for many processes, their calculation is still computationally very intensive, since reemissions or specular-like reflexions need to be considered. Three-dimensional simulations are, therefore, often limited to simplified models or small structures. A promising approach to overcome these limitations is a Monte Carlo technique, which simulates many particle trajectories in order to derive the corresponding flux distributions. In this work, a new technique to apply the Monte Carlo calculation directly to the implicit level set surface representation is presented. Furthermore, in order to minimize the computational costs of this approach, advanced ray tracing algorithms and data structures were applied. These methods have been adapted and optimized for the requirements of topography simulations.

Finally, the presented numerical methods were tested on various process models which have been reported in literature in order to demonstrate their wide applicability, especially for large three-dimensional structures.

Acknowledgement

First, I would like to thank my advisor Prof. Siegfried Selberherr for providing me with a perfect working environment, allowing me the freedom to pursue my own ideas, and giving me the opportunity to attend several international conferences.

I am also very grateful to my colleagues Johann Cervenka and Franz Schanovsky for their support concerning the computer cluster, Paul-Jürgen Wagner for helping me with many LaTeX issues, and Lado Filipović for proofreading this work.

Finally, I would like to thank Stefan Halama, who gave me the opportunity to gain a lot of experience during an internship with an impressive company.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BTRM	ballistic transport and reaction model
CFL	Courant-Friedrichs-Lewy
CMP	chemical-mechanical planarization
CPU	central processing unit
CVD	chemical vapor deposition
DC	direct current
DTG	dynamic tubular grid
EVRM	equi-volume rate model
FIB	focused ion beam
FWHM	full width at half maximum
H-RLE	hierarchical run-length encoded
IC	integrated circuit
KOH	potassium hydroxide
LS	level set
MC	Monte Carlo
MEMS	microelectromechanical systems
OM	object median
PVD	physical vapor deposition
RF	radio frequency
RLE	run-length encoded
SAH	surface area heuristic
SM	spatial median
STL	standard template library
TCAD	technology computer-aided design
TEOS	tetraethyl orthosilicate

List of Symbols

A	area
A_{ref}	reference area
$\vec{b}_{\max}$	maximum index vector of a rectangular axis aligned box
$\vec{b}_{\min}$	minimum index vector of a rectangular axis aligned box
$\mathcal{B}$	rectangular axis aligned box
C_{CFL}	CFL number
d	diameter
D	number of dimensions
D_i^-	backward difference operator
D_i^0	central difference operator
D_i^+	forward difference operator
$\mathcal{D}$	tangential disk
e	elementary charge $= 1.602 \times 10^{-19}\,\mathrm{C}$
$\vec{e}_i$	unit vector in x_i-direction
E	energy
F	flux on surface $\mathcal{S}$
F^{src}	flux on source plane $\mathcal{P}$
F^{tot}	total flux through source plane $\mathcal{P}$
$\vec{g}^{\max}$	maximum index vector of grid
$\vec{g}^{\min}$	minimum index vector of grid
G	reemission probability function
$\mathcal{G}$	set of all grid point index vectors, $\mathcal{G} \subseteq \mathbb{Z}^D$
H	Hamiltonian
$\hat{H}$	numerical approximation to the Hamiltonian
I	electric current
k_B	Boltzmann constant $= 1.381 \times 10^{-23}\,\mathrm{J\,K^{-1}}$
$\mathcal{L}$	layer of grid points, $\mathcal{L} \subseteq \mathcal{G}$
m	mass
M	number of different material regions
$\mathcal{M}$	material region
$\vec{n}$	normal unit vector on $\mathcal{S}$, $\vec{n} = (n_1, \ldots, n_D)$
$\vec{n}_{\mathcal{P}}$	normal unit vector on $\mathcal{P}$ pointing towards $\mathcal{S}$
N_{B}	number of boxes of the spatial subdivision
N_{CPU}	number of CPUs
N_{D}	number of defined grid points

N_E	number of edges in a graph
N_R	number of surface rates
N_P	number of simulated particles
N_V	number of vertices in a graph
$\mathcal{O}$	big O notation
$\vec{p}$	grid point indices, $\vec{p} = (p_1, \ldots, p_D) \in \mathbb{Z}^D$
P	pressure
$\mathcal{P}$	source plane
q	particle species
Q	number of process relevant particle species
r	radius
R	surface rate
s	sticking probability
$\mathcal{S}$	surface
t	time
T	temperature
$\bar{v}$	mean particle velocity
V	surface velocity (field)
w	particle weight factor
$\vec{x}$	point in space, $\vec{x} = (x_1, \ldots, x_D) \in \mathbb{R}^D$
$\vec{x}_\mathrm{cp}$	closest point on surface
Y	yield function
Y^tot	total sputter rate
Γ	arrival flux distribution on surface $\mathcal{S}$
Γ_re	reemitted flux distribution on surface $\mathcal{S}$
Γ_src	arrival flux distribution on source plane $\mathcal{P}$
δ	Dirac/Kronecker delta function
Δt	time increment
Δx	grid spacing
η	surface reaction order
θ	polar angle
Θ	surface coverage
κ	mean curvature
$\bar{\lambda}$	mean free path
ν	exponent in power cosine distribution
ρ	bulk density
σ	standard deviation
ϕ	azimuthal angle
χ	weight factor for SAH
$\vec{\omega}$	direction unit vector
Ω	solid angle

1 Introduction

Over the past 50 years, few developments have influenced human life more than the evolution of semiconductor technology. Electronic devices, such as computers or mobile telephones, which are now taken for granted, would have not been possible without the exponential technological progress experienced over decades, as proposed by Moore's Law [83]. The drive to reduce costs while increasing functionality has lead to the development of fabrication technologies that allow the production of devices with feature sizes under 100 nm. This enables the manufacture of integrated circuits (ICs) with billions of devices on a single chip.

The wide range of knowledge gained from the production of small structures had an impact also on fields other than electronics. The highly developed processes are used in a modified form to manufacture small sensors and actuators, generally known as microelectromechanical systems (MEMS).

1.1 Semiconductor Process Technology

Starting with a raw silicon wafer, many processing steps are necessary to obtain a final structure with the desired properties. The main processing steps used for ICs and MEMS manufacturing are listed in the following.

Lithography processing steps define patterns on the surface. The standard procedure transfers the mask information to the wafer surface, coated with a photoresist, using optical imaging. A final development step removes exposed regions of the photoresist.

Etching processes are used to remove material from the wafer. One can distinguish between wet and dry etching processes using liquid or gaseous etching agents, respectively. The first are characterized by isotropic uniform or crystallographic direction dependent etch rates, while the latter also allow directional etching. Therefore, to transfer patterns defined by the photoresist onto the wafer surface, dry etching processes are mainly used today.

Deposition of new material layers is essential for creating insulators, conductors, or n- and p-type semiconductors. It is possible to categorize deposition processes into physical vapor deposition (PVD) and chemical vapor deposition (CVD) processes depending on the primary mechanism used to add a material to the surface. In

the first case, this is condensation of the corresponding vaporized material. In the second case, the underlying mechanism is a chemical reaction at the surface.

Chemical-mechanical planarization is used to flatten an irregular wafer surface. The combination of a chemical slurry with mechanical polishing allows for a very accurate planarization of the surface. This is important for photolithography, where the entire wafer surface needs to be within the depth of field of the optical imaging system.

Oxidation converts silicon to silicon dioxide on top of the surface using oxygen (dry oxidation) or water (wet oxidation). Silicon layers are used as insulators, as masks, or as scattering layers for ion implantation. To grow silicon dioxide layers on top of non-silicon layers tetraethyl orthosilicate (TEOS) deposition or silane pyrolysis are used instead.

Ion implantation is the common technique to introduce dopants into the semiconductor. High energy ions are able to penetrate into the target. The penetration depth can be controlled by the ion energy and the incidence angle. Subsequent annealing allows for the redistribution of dopants and repairs lattice defects caused by ion bombardment.

Diffusion of dopants from the wafer surface into the semiconductor is another technique for doping. The dopants can be supplied from a gas or liquid phase, or from a pre-deposited layer. Diffusion within the semiconductor is mainly controlled by temperature and is often a parasitic effect of subsequent processing steps which operate at higher temperatures.

Focused particle beams are generated by accelerating ions or electrons towards the wafer surface and focusing them using an electromagnetic imaging system similar to that of electron microscopes [133]. Beam diameters down to 10 nm are possible and can be used for structuring either by local removal or deposition of material. The small throughput of particle beam processes is not feasible for series production of ICs. However, they are useful for sample sectioning, mask repair, or MEMS fabrication [99, 132].

1.2 Technology Computer-Aided Design

The short product development cycle in today's semiconductor industry would not have been possible without the heavy support of computational resources. Technology computer-aided design (TCAD) plays a crucial role in semiconductor product development. TCAD aims to model the entire semiconductor process flow, as well as device and circuit operation. Such simulations are essential to obtain a better understanding of the underlying physics. Moreover, if appropriate physical models, together with numerical schemes solving the governing equations, are available, process flows or device designs can be optimized on the computer instead of doing time- and cost-intensive experiments in the real world.

An interesting aspect in the field of TCAD is that the complexity of some problems scales with the currently available computation power. This is in contrast to other technical fields, where it is acceptable to wait for faster central processing units (CPUs) to make a problem solvable. However, this is not possible in the semiconductor industry, since TCAD addresses the design of next generation CPUs.

As a consequence, TCAD is a demanding field always hindered by the constraint given by available computational resources. Hence, the development and usage of fast and well scaling algorithms is essential for TCAD in order to counter the increasing problem sizes.

1.3 Motivation

This work continues the work at the Institute for Microelectronics at the Vienna University of Technology on the simulation of semiconductor processes which alter the topography, such as etching and deposition processes. The first topography simulator developed at the institute by Strasser [122, 123] and Pyka [93] uses a structuring element algorithm. A better approach is the level set (LS) method [86], which has been implemented by Heitzinger within the ELSA framework [40]. Sheikholeslami extended this framework to three dimensions and used it for the Topo3D simulator [111]. Due to a poor scaling behavior with problem size, Topo3D reaches its computational limits easily, when calculating the particle transport for advanced physical models on larger structures. Furthermore, it is not able to describe etching processes on structures with multiple materials. The goal of this work was to investigate, improve, and develop algorithms which are convenient to overcome the shortcomings of previous topography simulators.

1.4 Outline of the Thesis

A general prerequisite for computer simulations is the existence of an appropriate mathematical model describing the physical behavior. Therefore, an overview of common models used for topography simulations is provided in Chapter 2. After justifying the usage of a continuum model, models for particle transport as well as for surface kinetics are discussed. Generalized equations are presented, which are able to cover a large variety of different process models.

From the numerical point of view, two techniques are necessary to perform topography simulations. First, a method is needed which is able to describe the evolving geometry over time. This is the topic of Chapter 3, where an overview of different approaches including their basic ideas are given at the beginning. The main part of this chapter deals with the LS method which is used throughout this work. Chapter 4 continues

thematically with the LS method and gives a detailed description of a fast LS framework which was developed and implemented using modern LS algorithms and data structures, enhanced by the capability of handling multiple material regions. The last part of this chapter describes the parallelization on shared memory computers.

The LS method requires the knowledge of the surface velocities in order to calculate the deformation of the geometry over time. Hence, a second method is required which efficiently calculates the required surface rates by solving the corresponding particle transport and surface reaction equations of the implemented model. The solution of the generalized model equations, as introduced in Chapter 2, is the topic of Chapter 5. There, Monte Carlo (MC) simulations are proposed to solve the transport equations and to overcome some limitations of the conventional direct integration technique. For this MC approach a well behaved scaling law can be obtained, if modern ray tracing techniques are used, which are discussed in more detail and which are optimized for the needs of topography simulation.

All the numerical techniques are finally demonstrated in Chapter 6 for a selection of different processes. Among the examples are simulations of CVD, plasma etching, anisotropic wet etching, the Bosch process, and focused ion beam (FIB) processing. Finally, Chapter 7 concludes with a brief summary and gives some ideas for future work.

2 Process Modeling

The main focus of this work is put on three-dimensional simulations of a topography changing process, especially for the simulation of thin film deposition and etching with particle transport in rarefied gas phases. This chapter attempts to integrate a wide range of popular models within a generalized version of the ballistic transport and reaction model (BTRM) [21]. In the following section, physical models for the description of the solid body, the particle transport to the surface, as well as the surface reaction are discussed.

2.1 Continuum Approach

Since the aim of semiconductor processing is the modification of the wafer surface, an appropriate model to describe the solid body is required. The natural way is an atomistic description, where the process is simulated using cellular automata or Monte Carlo techniques [13, 33, 47, 78, 118]. These atomistic approaches are able to predict characteristics as a microstructure or as surface roughness. However, the atomistic treatment is computationally very intensive and is therefore limited to small structures.

If the main concern is the prediction of the final shape of the structure, and not the microscopic properties, and if the typical structure sizes are much larger than the typical lattice constant, the solid body can be described as a continuum. The continuum approach mathematically represents the solid body as a region $\mathcal{M} \in \mathbb{R}^D$ with the surface $\mathcal{S} = \partial\mathcal{M}$ as its boundary. Here D is the number of dimensions. Material deposition or removal is simply described by moving the interface $\mathcal{S}$ with respect to the deposition or removal rates on the surface.

Subsequent process steps often result in a multi-layer structure consisting of different material types. A common assumption within the continuum approach is that process relevant properties are homogeneous within each material region. Furthermore, the transition between two material layers is assumed to be abrupt. Hence, the entire geometry $\mathcal{M}$ can be described by M disjoint material regions $\mathcal{M}_i$ which fulfill

$$\mathcal{M} = \bigcup_{i=1}^{M} \mathcal{M}_i \quad \text{and} \quad \mathcal{M}_i \cap \mathcal{M}_j = \emptyset \text{ for all } i \neq j. \tag{2.1}$$

As a consequence, knowing the surface and the interfaces between the different material regions is sufficient to describe the different material regions.

2.2 Transport Kinetics

In dry processes, where the wafer is exposed to a gas phase, the flux distribution of particles on the surface is usually the crucial factor for the local deposition or etching rates. Common models for the particle transport within the process chamber to the wafer surface divide the gas phase region by a plane $\mathcal{P}$ just above the surface $\mathcal{S}$ as depicted in Figure 2.1 [32, 80]. A first model describes the transport within the reactor-scale region to $\mathcal{P}$ and allows the determination of the flux distribution

$$\Gamma_{\text{src}} = \Gamma_{\text{src}}(\vec{x}; q, \vec{\omega}, E) \quad \text{with } \vec{x} \in \mathcal{P} \tag{2.2}$$

which describes the number of arriving particles of species q with incident direction $\vec{\omega}$ and energy E per unit area. Here $\vec{\omega}$ is a unit vector. The total flux F_q^{src} of particles of type q crossing $\mathcal{P}$ is the integral over all directions and energies

$$F_q^{\text{src}}(\vec{x}) = \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}_{\mathcal{P}}>0} \Gamma_{\text{src}}(\vec{x}; q, \vec{\omega}, E)\, d\Omega\, dE. \tag{2.3}$$

Here $\vec{n}_{\mathcal{P}}$ denotes the normal vector to $\mathcal{P}$ pointing towards $\mathcal{S}$ (see Figure 2.1).

A second model describes the continued particle transport from $\mathcal{P}$ to $\mathcal{S}$ within the feature-scale region using (2.2) as a boundary condition. The advantage of dividing the gas phase region is that the particle transport acts on different length scales, and hence, different transport models can be applied to each region.

2.2.1 Reactor-Scale Transport

Although process simulation in the reactor-scale region is not the scope of this work, some basic considerations of different transport models and typical characteristics of flux distribution on $\mathcal{P}$ are given in this section. The transport of particles can be characterized by the mean free path $\bar{\lambda}$. For an ideal gas $\bar{\lambda}$ calculates as [22]

$$\bar{\lambda} = \frac{k_B T}{\sqrt{2}\pi d^2 P}. \tag{2.4}$$

Here $k_B = 1.381 \times 10^{-23}\,\text{J K}^{-1}$ denotes the Boltzmann constant, T is the temperature, P is the pressure, and d is the collision diameter of a gas molecule, which is about 0.4 nm for most molecules of interest [91].

If the mean free path is much smaller than the representative physical length scale, which is true in the case of the reactor dimension, the transport is said to be in the

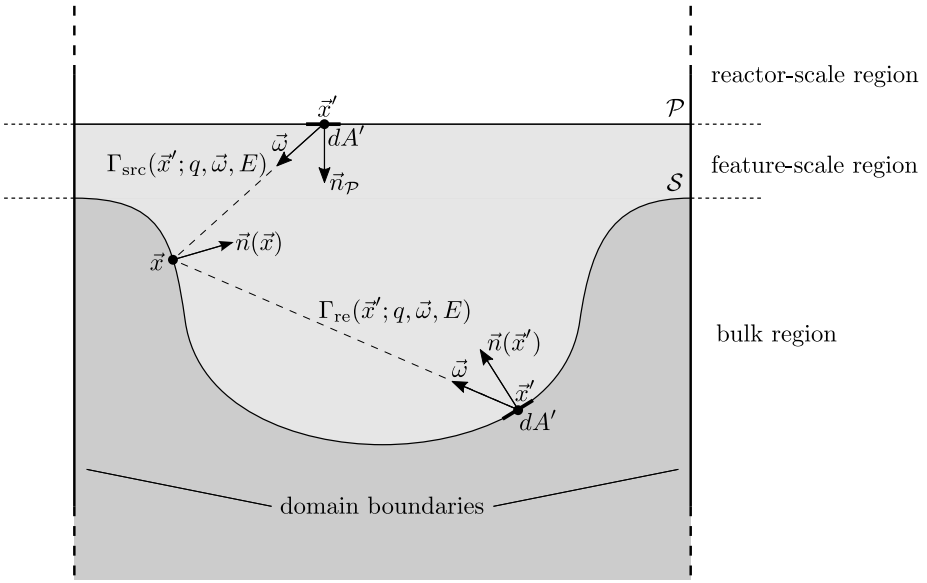


Figure 2.1: The particle transport is broken up by describing the transport to $\mathcal{P}$ on reactor-scale and the transport from $\mathcal{P}$ to $\mathcal{S}$ on feature-scale.

continuum regime [32, 80]. This is usually the case for CVD and dry etching processes with typical pressures in the range 1 Pa to 100 Pa and a typical temperature of 500 K resulting in a mean free path in the range 100 µm to 10 mm. Hence, the velocities of a neutral particle species q_{neu} can be assumed to follow the Maxwell–Boltzmann distribution, leading to a cosine-like directional dependence of the flux distribution on $\mathcal{P}$

$$\Gamma_{\mathrm{src}}(\vec{x};q_{\mathrm{neu}},\vec{\omega},E) = F_{\mathrm{neu}}^{\mathrm{src}}\frac{1}{\pi}\cos\theta \quad \text{with } \cos\theta = \vec{\omega}\cdot\vec{n}_{\mathcal{P}}. \tag{2.5}$$

Here the energy distribution is neglected, because the kinetic energy for typical process temperatures is usually too small to play a role in the surface reaction. θ is the angle between the incident direction $\vec{\omega}$ and $\vec{n}_{\mathcal{P}}$. $F_{\mathrm{neu}}^{\mathrm{src}}$ is the flux on $\mathcal{P}$ and can be assumed constant for all $\vec{x}\in\mathcal{P}$ at feature-scale. The value for $F_{\mathrm{neu}}^{\mathrm{src}}$ can either be obtained by calculating the reactor-scale transport or experimentally by measurements.

In many processes plasmas are used, leading to the need to model ion transport. Ions are accelerated towards the wafer surface due to the plasma sheath potential. This leads to a narrow angle distribution often described as power cosine distribution [91], also known as Phong distribution in the field of computer graphics [90]

$$\Gamma_{\mathrm{src}}(\vec{x};q_{\mathrm{ion}},\vec{\omega},E) \sim (\cos\theta)^{\nu}. \tag{2.6}$$

For large exponents ν this is equivalent to a normal distribution

$$(\cos\theta)^{\nu} \approx \exp(-\beta\theta^2) \quad \text{with } \nu = 2\beta \gg 1. \tag{2.7}$$

Together with the energy distribution $\epsilon_{\text{ion}}(E)$ where $\int_0^\infty \epsilon_{\text{ion}}(E)\,dE = 1$ the arrival flux distribution can be expressed as

$$\Gamma_{\text{src}}(\vec{x}; q_{\text{ion}}, \vec{\omega}, E) = F_{\text{ion}}^{\text{src}} \frac{1+\nu}{2\pi} (\cos\theta)^\nu \, \epsilon_{\text{ion}}(E). \tag{2.8}$$

If no collisions occur in the sheath, and if the final energy of ions is much larger than the initial random thermal energy, $E \gg k_B T$, the cosine exponent ν can be estimated by [34]

$$\nu = \frac{2E}{k_B T}. \tag{2.9}$$

For direct current (DC) biased plasmas the final ion energy can be assumed to be monoenergetic with $E = eV_{\text{sheath}}$, where $e = 1.602 \times 10^{-19}\,\text{C}$ denotes the elementary charge and V_{sheath} is the sheath potential. Hence, the energy distribution can be approximated by a Dirac delta function

$$\epsilon_{\text{ion}}(E) = \delta(E - eV_{\text{sheath}}). \tag{2.10}$$

Radio frequency (RF) biased plasmas result in more complex energy distributions with a characteristic bimodal shape. Plasma sheath simulations using MC techniques are able to predict the energy distribution function and can also incorporate ion collisions in the plasma sheath region [27, 56].

For most processes the flux distribution Γ_{src} does not vary significantly over $\mathcal{P}$ at feature-scale. However, this is not true for processes working with local particle bombardments such as FIBs. The spatial intensity distribution of such a beam is usually described by a Gaussian profile [57, 132, 133]

$$\Gamma_{\text{src}}(\vec{x}; q_{\text{ion}}, \vec{\omega}, E) \sim \frac{I}{e} \cdot \frac{1}{2\pi\sigma^2} \exp\left(-\frac{r^2}{2\sigma^2}\right), \tag{2.11}$$

where I denotes the beam current with typical values in the range 1 pA to 10 nA [99] and r is the distance from the center line of the beam. σ is proportional to the beam diameter d, which is commonly defined as the full width at half maximum (FWHM), thus $\sigma = d\frac{1}{2\sqrt{2\ln 2}}$.

At feature-scale it is a reasonable approximation to regard the beam as monoenergetic and unidirectional. If E_0 is the ion beam energy, typically ranging from 10 kV to 50 kV [99], and if $\vec{\omega}_0$ represents the incidence direction, the arrival angular energy flux distribution can be written as

$$\begin{aligned}\Gamma_{\text{src}}(\vec{x}; q_{\text{ion}}, \vec{\omega}, E) = \\ \frac{I}{e} \cdot \frac{\vec{\omega}_0 \cdot \vec{n}_{\mathcal{P}}}{2\pi\sigma^2} \exp\left(-\frac{\|\vec{x} - \vec{x}_0\|^2 - (\vec{\omega}_0 \cdot (\vec{x} - \vec{x}_0))^2}{2\sigma^2}\right) \delta^2(\vec{\omega} - \vec{\omega}_0)\delta(E - E_0).\end{aligned} \tag{2.12}$$

Here δ^2 denotes a two-dimensional delta distribution satisfying $\int f(\vec{\omega})\delta^2(\vec{\omega} - \vec{\omega}_0)\,d\Omega = f(\vec{\omega}_0)$.

2.2.2 Feature-Scale Transport

Plane $\mathcal{P}$ together with surface $\mathcal{S}$ represent the boundary of the feature-scale region (see Figure 2.1). If the arriving flux distribution Γ_{src} is known at $\mathcal{P}$, and for all surface points $\vec{x} \in \mathcal{S}$ the reemitted flux distribution $\Gamma_{\text{re}} = \Gamma_{\text{re}}(\vec{x}; q, \vec{\omega}, E)$ is given, the particle transport through the feature-scale region is well-defined. This allows the determination of the arriving angular energy particle flux distribution $\Gamma = \Gamma(\vec{x}; q, \vec{\omega}, E)$ on the surface $\mathcal{S}$, which principally defines the surface rates. The dependence of the reemitted flux distribution Γ_{re} on the local arrival flux distribution Γ will be discussed later.

For most processes the frequency of particle–particle collisions at feature-scale is negligible relative to particle–surface collisions [21]. For comparison, according to (2.4) the mean free path for an ideal gas at room temperature ($T \approx 300\,\text{K}$) and atmospheric pressure ($P \approx 100\,\text{kPa}$) is approximately 60 nm, which is of the same magnitude as typical sizes of modern structures. However, most dry processes operate at much lower pressures, where the mean free path is much larger than typical feature sizes. This validates the assumption of ballistic transport within the feature-scale region.

The average particle velocities are usually much larger than the surface rates. Kinetic theory gives the mean particle velocity $\bar{v}$ of an ideal gas as [22]

$$\bar{v} = \sqrt{\frac{8 k_B T}{\pi m}}. \tag{2.13}$$

For typical temperatures T and molecular weights m the average velocity is larger than $100\,\text{m}\,\text{s}^{-1}$. In comparison, the surface rates in most processes are well below $1\,\mu\text{m}\,\text{s}^{-1}$. As a consequence, the surface can be regarded to be constant and the travel time of particles can be neglected for calculating the arrival flux distribution at a certain time.

In the ballistic transport regime the particle trajectories of neutral particles are straight lines. For charged particles, such as ions, electromagnetic forces must be incorporated. Incident ions charge insulating layers, such as SiO_2. This leads to a static electric field influencing the ion trajectories [30, 48]. If the electric field becomes strong enough, the trajectories can be disturbed significantly, which affects the final profile. For example, charging is an essential mechanism for the notching effect which can be observed, if polysilicon-on-insulator structures are overetched [48]. Despite the importance of charging for the description of some effects, this work does not incorporate electromagnetic forces for the intra-feature particle transport. However, as will be outlined in Chapter 7 the main ideas of this work can be extended to incorporate electrostatic fields.

In the absence of electromagnetic forces the trajectories of all particles are straight lines. The particles which arrive at a surface point $\vec{x} \in \mathcal{S}$ with incident direction $\vec{\omega}$

can either originate from $\mathcal{P}$ or from the surface itself due to reemission, as shown in Figure 2.1. Thus, the particle transport at feature-scale can be described by

$$\Gamma(\vec{x};q,\vec{\omega},E)\,d\Omega = \begin{cases} \dfrac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\Gamma_{\mathrm{src}}(\vec{x}';q,\vec{\omega},E)\,dA' & \text{if } \vec{x}'\in\mathcal{P}, \\ \dfrac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\Gamma_{\mathrm{re}}(\vec{x}';q,\vec{\omega},E)\,dA' & \text{if } \vec{x}'\in\mathcal{S}. \end{cases} \tag{2.14}$$

Here $\vec{x}'\in\mathcal{P}\cup\mathcal{S}$ is the point seen from point $\vec{x}$ in direction $-\vec{\omega}$, thus $\vec{\omega}=\frac{\vec{x}-\vec{x}'}{\|\vec{x}-\vec{x}'\|}$. dA' is an infinitesimal surface element of $\mathcal{S}$ or $\mathcal{P}$ around $\vec{x}'$. $\vec{n}(\vec{x})$ is the surface normal at point $\vec{x}$.

2.2.3 Reemission

For the determination of the arrival flux distribution Γ the reemitted flux distribution Γ_{re} must be known. This work assumes that the reemission is related to the incident flux distribution by

$$\Gamma_{\mathrm{re}}(\vec{x};q,\vec{\omega},E) = \sum_{q'=1}^{Q}\int_0^\infty \int_{\vec{\omega}'\cdot\vec{n}(\vec{x})<0} G(\vec{x};q,\vec{\omega},E;q',\vec{\omega}',E')\Gamma(\vec{x};q',\vec{\omega}',E')\,d\Omega'\,dE'. \tag{2.15}$$

$G(\vec{x};q,\vec{\omega},E;q',\vec{\omega}',E')$ is the reemission probability function which describes the conditional probability for the emission of a particle of species q and energy E into direction $\vec{\omega}$, if a particle of species q', direction $\vec{\omega}'$, and energy E' strikes the surface at point $\vec{x}\in\mathcal{S}$. Q is the number of process relevant particle species. This model assumes that the time between the incidence and the induced reemissions is negligible. A further assumption is that the new particles are reemitted exactly from the same location, where the incident particle has hit the surface.

The description, using the reemission probability function, is quite general in the sense that many effects can be properly treated. For example, perfect reflection of particles of type q would be described by

$$G(\vec{x};q,\vec{\omega},E;q,\vec{\omega}',E') = \delta(E-E')\delta^2(\vec{\omega}-\vec{\omega}'+2\cdot\vec{n}(\vec{x})\cdot(\vec{n}(\vec{x})\cdot\vec{\omega}')). \tag{2.16}$$

A more realistic description of specular-like reflexions of ions is given in [15]. If the incident direction is given by the azimuthal and the polar angle $\vec{\omega}'=(\phi',\theta')$ relative to the surface normal $\vec{n}$ with $\phi'\in[0,2\pi[$ and $\theta'\in\left]\frac{\pi}{2},\pi\right]$, then the average reemitted direction is $\vec{\omega}_{\mathrm{avg}}=(\phi_{\mathrm{avg}},\theta_{\mathrm{avg}})=(\phi',\min(\pi-\theta',\theta_{\max}))$. Here $\theta_{\max}$ is a constant parameter smaller than, but close to $\frac{\pi}{2}$. The reemitted direction $\vec{\omega}$ is distributed around $\vec{\omega}_{\mathrm{avg}}$ according to

$$G(\vec{x};q,\vec{\omega},E;q',\vec{\omega}',E')\sim\begin{cases}\cos\left(\frac{\pi}{2}\frac{\theta}{\frac{\pi}{2}-\theta_{\mathrm{avg}}}\right) & \text{if } \theta\le\frac{\pi}{2}-\theta_{\mathrm{avg}}, \\ 0 & \text{else,}\end{cases} \tag{2.17}$$

with $\theta=\arccos(\vec{\omega}\cdot\vec{\omega}_{\mathrm{avg}})$.

Diffusive reemission of a neutral particle species q_{neu} is usually described by the relation

$$G(\vec{x}; q_{\text{neu}}, \vec{\omega}, E; q_{\text{neu}}, \vec{\omega}', E') = (1-s)\,\frac{1}{\pi}\cos\theta \quad \text{with } \cos\theta = \vec{n}\cdot\vec{\omega}. \tag{2.18}$$

Here s is the probability that an incident particle remains sticking on the surface. The directional distribution of diffusive reemitted particles follows the Knudsen cosine law [37]. Since the total kinetic energy of neutrals is usually given only by their thermal energy, the energy distribution of neutrals can be neglected.

The incidence of high energetic ions can lead to sputtering of particles on the surface, which are redeposited somewhere else. For example, if q_{ion} denotes ions and q_{sp} denotes a sputtered particle type for which the kinetic energy can be neglected and a cosine-like directional distribution can be assumed, the reemission probability function takes the form [60]

$$G(\vec{x}; q_{\text{sp}}, \vec{\omega}, E; q_{\text{ion}}, \vec{\omega}', E') = Y(\theta', E')\cdot\frac{1}{\pi}\cos\theta. \tag{2.19}$$

Here $Y(\theta', E')$ is a yield function describing the average number of sputtered particles per incidence of an ion at angle θ' and energy E'. Commonly, yield functions are approximated by

$$Y(\theta, E) \sim f(\theta)\cdot\max\left(0, \sqrt{E} - \sqrt{E_{\text{th}}}\right) \tag{2.20}$$

This reflects the observed linear growth with the root of the incident energy E, which must be larger than the threshold energy E_{th} to permit sputtering [120]. The angular dependence is usually obtained by measurements. A semi-empirical formula was proposed by Yamamura et al. [76, 136]

$$f(\theta) = (\cos\theta)^{-C_1}\cdot\exp\left(C_2\cdot\left(1-(\cos\theta)^{-1}\right)\right). \tag{2.21}$$

Here C_1 and C_2 are positive fitting parameters. For $C_1 > C_2$ this function exhibits a maximum at inclined incidence and is decreasing towards zero for $\theta \to \frac{\pi}{2}$.

In general, the reemission probability function may itself depend on the flux distribution

$$G = G(\vec{x}; q, \vec{\omega}, E; q', \vec{\omega}', E'; \Gamma). \tag{2.22}$$

For example, the sticking probability in (2.18) can depend on the so-called surface coverage which describes the fraction of occupied surface sites. The coverage may be influenced by the fluxes of inhibitors which adsorb on the surface and lead to some passivation of the surface. If the reemission probability function shows a dependence on the flux distributions, (2.15) will be non-linear and the calculation of the particle transport will get more complicated. This issue will be addressed in further detail in Section 5.4.1.

Finally, it should be noted that in the presence of different material regions the reemission probabilities are also dependent on the material type on the surface. As an

example, the type and the number of sputtered particles vary with the material on the surface. Therefore, for each material type exposed during the process, an individual reemission probability function must be defined.

2.3 Surface Kinetics

Once the particle transport to the surface is computed, the etch or deposition rate can be determined, which gives the desired surface velocity. In this work the arrival flux distributions are assumed to be in a pseudo-steady state. They only depend on the surface profile and the arrival flux distribution at plane $\mathcal{P}$ and not on their history. The surface velocity $V(\vec{x})$ with $\vec{x} \in \mathcal{S}$ can be written as a functional of the local arriving flux distribution $\Gamma(\vec{x})$

$$V(\vec{x}) = V(\Gamma(\vec{x})). \tag{2.23}$$

For the numerical representation of the continuous flux distribution Γ some form of discretization is required. If the functions $r_l(\vec{x}; q, \vec{\omega}, E)$ with $1 \leq l \leq N_\mathrm{R}$ are appropriate weight functions which map $\Gamma(\vec{x})$ to a finite number of scalar values given by

$$R_l(\vec{x}) := \sum_{q=1}^{Q} \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} r_l(\vec{x}; q, \vec{\omega}, E)\Gamma(\vec{x}; q, \vec{\omega}, E)\, d\Omega\, dE, \tag{2.24}$$

the surface velocity can be written as a function of these scalars

$$V(\vec{x}) = V(R_1(\vec{x}), R_2(\vec{x}), \ldots, R_{N_\mathrm{R}}(\vec{x})). \tag{2.25}$$

To save memory it is necessary to minimize the number of discretized values N_R that need to be stored for each surface point. In Section 5.2 a method will be presented which does not require a discretization of the flux distribution in order to calculate the particle transport. Hence, the choice of weight functions must be appropriate primarily for an accurate calculation of the surface velocity.

Luckily, most models require only a minimum number of weight functions r_l. Typically, all the information needed from the flux distribution Γ for a certain particle type can be mapped to a single value representing a certain rate using a single weight function. This is not surprising, since only the total flux, in case of neutrals, or the effective sputter yield, in case of high energy ions, is required for common process models. The total flux $F_{q'}(\vec{x})$ of particles of species q' is obtained by choosing $r_l(\vec{x}; q, \vec{\omega}, E) := \delta_{qq'}$ which yields

$$R_l(\vec{x}) = \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} \Gamma(\vec{x}; q', \vec{\omega}, E)\, d\Omega\, dE = F_{q'}(\vec{x}) \tag{2.26}$$

For particles with higher kinetic energies, which are generally ions, the total flux is of less interest. The contribution of such particles to the surface velocity depends on

the incident angle θ and energy E and is described by a yield function $Y(\theta, E)$ as introduced in Section 2.2.3. The total sputter rate $Y_{q'}^{\text{tot}}(\vec{x})$ for particles of species q' can be obtained by choosing $r_l(\vec{x}; q, \vec{\omega}, E) := \delta_{qq'} Y(\theta, E)$

$$R_l(\vec{x}) = \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} \Gamma(\vec{x}; q', \vec{\omega}, E) Y(\theta, E)\, d\Omega\, dE = Y_{q'}^{\text{tot}}(\vec{x}). \tag{2.27}$$

Generally, all the R_l values represent a rate on the surface, such as flux or total sputter rate of a certain particle species. Therefore, from now on, they will be interpreted as surface rates. The surface velocity V is a function of these surface rates.

2.3.1 Linear Surface Reactions

In many models the surface velocity is simply written as a linear function of rates

$$V(\vec{x}) = \sum_{l=1}^{N_{\text{R}}} C_l \cdot R_l(\vec{x}), \tag{2.28}$$

where C_l are constant coefficients. If this is the case, the individual contributions can be combined into a single rate R which is obtained by taking the superposition of all weight functions r_l as the new weight function

$$r(\vec{x}; q, \vec{\omega}, E) := \sum_{l=1}^{N_{\text{R}}} C_l \cdot r_l(\vec{x}; q, \vec{\omega}, E). \tag{2.29}$$

The surface velocity can then be written as

$$V(\vec{x}) = R(\vec{x}). \tag{2.30}$$

Simple deposition and etching models often assume such a linear surface reaction. If only a single particle species dominates the surface reaction, the surface velocity is typically modeled as

$$V(\vec{x}) = \frac{1}{\rho} \cdot \Delta m \cdot s \cdot F(\vec{x}). \tag{2.31}$$

Here F is the total incident flux of particles, s is the sticking or reaction probability, Δm is the mass which is deposited ($\Delta m > 0$) or removed ($\Delta m < 0$) from the surface per reacting incident particle, and ρ is the bulk density [91]. Advanced models use multiple particle types. For example, a linear model for TEOS deposition is given in [51]. There, different reaction paths are assumed. TEOS is modeled to be either deposited directly with a small sticking coefficient or through deposition of a very reactive intermediate with a large sticking coefficient, that is formed by gas phase reactions. Linear surface reactions are also used for some simple etching models, which require the incorporation of physical etching (sputtering) as well as chemical

etching. If the two components are independent from each other, the reaction can be described by (2.28) [91].

Processes operating in the transport-controlled regime, which means that the particle transport to the surface is the crucial limiting factor for the surface velocity, can often be described by linear reaction models. There the surface velocity is essentially proportional to the particle transport to the surface.

2.3.2 Non-Linear Surface Reactions

In general, the surface velocity (2.25) is not a linear function. As an example, the sticking probability in (2.31) can depend on the local arriving particle flux, if higher order surface kinetics are assumed [21]

$$s(\vec{x}) \sim F(\vec{x})^{\eta-1}. \tag{2.32}$$

Here η is the order of the reaction. Obviously, in the case of non-linear reaction kinetics ($\eta \neq 1$) the sticking coefficient depends on the flux.

Other examples, where non-linear surface reactions need to be incorporated, are processes with ion-enhanced etching. There physical (ions) and chemical (neutrals) components act in a synergistic manner, so that the etch rate is larger than that obtained by summing up their individual contributions.

The Langmuir adsorption model is able to capture this behavior [91]. The idea is to assume an absorbed state of byproducts. The fraction of surface sites covered by these byproducts is called the surface coverage $\Theta = \Theta(\vec{x})$. The etch rate is then composed of three contributions

$$V(\vec{x}) = \underbrace{-\alpha_{\mathrm{ch}} \cdot \Theta(\vec{x})}_{\text{chemical etching}} \quad \underbrace{-\alpha_{\mathrm{ph}} \cdot Y_{\mathrm{ph}}^{\mathrm{tot}}(\vec{x})}_{\text{physical sputtering}} \quad \underbrace{-\alpha_{\mathrm{ie}} \cdot \Theta(\vec{x}) \cdot Y_{\mathrm{ie}}^{\mathrm{tot}}(\vec{x})}_{\text{ion-enhanced etching}}. \tag{2.33}$$

The first term corresponds to chemical etching which is proportional to the coverage, the second term represents physical sputtering with the total sputter rate $Y_{\mathrm{ph}}^{\mathrm{tot}}(\vec{x})$, and the last term is due to ion-enhanced etching, which is proportional to the coverage and the total ion-enhanced etching rate $Y_{\mathrm{ie}}^{\mathrm{tot}}(\vec{x})$. The total yields are both calculated using a weight function similar to that used in (2.27). Two different total yields are introduced, since in the general case the physical sputter yield and the ion-enhanced etching yield are not equal. The constants α_{ch}, α_{ph}, and α_{ie} in (2.33) are model parameters.

For the coverage $\Theta(\vec{x})$ a balanced equation can be set up as follows

$$\frac{d\Theta}{dt}(\vec{x}) = \underbrace{\beta_{\mathrm{ad}} \cdot (1 - \Theta(\vec{x})) \cdot F(\vec{x})}_{\text{adsorption}} \quad \underbrace{-\beta_{\mathrm{ch}} \cdot \Theta(\vec{x})}_{\text{chemical etching}} \quad \underbrace{-\beta_{\mathrm{ie}} \cdot \Theta(\vec{x}) \cdot Y_{\mathrm{ie}}^{\mathrm{tot}}(\vec{x})}_{\text{ion-enhanced etching}}. \tag{2.34}$$

The first term describes the adsorption of chemical components, which is proportional to the total arriving flux $F(\vec{x})$ of neutrals and the fraction of empty surface sites $(1 - \Theta(\vec{x}))$. The second and the third term are losses due to chemical and ion-enhanced etching, respectively, which are both proportional to the coverage. The constants β_{ad}, β_{ch}, and β_{ie} are again model parameters.

A common approach is to assume that the coverage is always in a steady state $\frac{\partial \Theta}{\partial t}(\vec{x}) = 0$. Therefore, the coverage can be explicitly expressed as a function of the rates $F(\vec{x})$ and $Y_{\text{ie}}^{\text{tot}}(\vec{x})$,

$$\Theta(\vec{x}) = \frac{\beta_{\text{ad}} F(\vec{x})}{\beta_{\text{ad}} F(\vec{x}) + \beta_{\text{ch}} + \beta_{\text{ie}} Y_{\text{ie}}^{\text{tot}}(\vec{x})}, \tag{2.35}$$

and can be plugged into (2.33). The result is a non-linear function of the rates for the surface velocity.

2.3.3 Transport-Independent Surface Reactions

The surface velocity can sometimes be assumed to be independent of any particle transport, if the process is reaction-limited, where enough reactants are always on the surface. This is generally the case for wet processes for which the diffusive transport to the surface can be neglected. The simplest case is isotropic etching with a constant etch rate.

The description of anisotropic wet etching is more complex, since the etch rate can depend on the crystallographic direction [96]. In this case the surface velocity depends only on the surface orientation described by the normal direction $\vec{n}(\vec{x})$

$$V(\vec{x}) = V(\vec{n}(\vec{x})). \tag{2.36}$$

3 Surface Evolution

The simulation of the topography changing processes requires a method capable to describe geometric deformations over time. To observe the final profile it is primarily important to track the surface $\mathcal{S}(t)$ over time. The initial surface $\mathcal{S}(t=0)$ can be extracted from the initial geometry ($\mathcal{S} = \partial\mathcal{M}$). During topography simulations the surface velocities $V(\vec{x})$ in normal direction are calculated at the surface points $\vec{x} \in \mathcal{S}(t)$. To obtain the final profile after the complete process time, t_{process}, the following problem must be solved:

- Given: $\mathcal{S}(t=0)$ and $V(\vec{x})$ for all $\vec{x} \in \mathcal{S}(t)$ and $t \in [0, t_{\text{process}}]$,
- Required: $\mathcal{S}(t_{\text{process}})$.

This chapter first gives an overview of different numerical methods for tracking moving boundaries. The pros and cons of these techniques are briefly discussed. Finally the focus is placed on the LS method which is probably the most popular method used in modern topography simulators, especially for three dimensions.

3.1 Boundary Evolution Techniques

The various approaches used to solve the boundary tracking problem can be classified into three categories. For each of them the basic concepts are presented in the following sections.

3.1.1 Segment-Based Methods

Segment-based methods explicitly describe a boundary. This means that a boundary in two- or three-dimensional space is usually represented as line segmentation [38, 52] or as triangulation [12, 31, 59, 68, 105, 131], respectively. The time evolution of the boundary is computed by advancing all nodes in the normal direction according to the given surface velocities at every time step. Since the surface normal is not defined at the nodes, it must be obtained by averaging the normals of neighboring segments.

The drawback of segment-based methods is that accumulation or rarefaction of nodes can occur, especially at regions with high curvature or large differences in surface velocities. Therefore, nodes must be deleted or inserted to keep their density, and consequently the surface resolution, appropriate. A further problem is the proper

treatment of self-intersections, which can occur if, for example, two different parts of the boundary merge. Hence, much effort is necessary to obtain an efficient and, more importantly, a robust boundary tracking algorithm, especially in three dimensions.

3.1.2 Cell-Based Methods

Cell-based methods use a grid of cubic cells to represent the geometry. Numerical values assigned to cells describe their contents. One can distinguish between methods using discrete and continuous values.

In the first case integral values are used to mark vacuum cells or cells with same properties of matter [26, 42, 93, 118, 122, 123]. To calculate the time evolution of the surface the state of the cells near the surface must be successively modified in consideration of the given surface rates. The cell removal technique, which was used to simulate photo resist development, determines the time required to completely remove each surface material cell [26, 42]. Iteratively, the state of the cell with the smallest removal time is changed, and the removal times are recalculated for all surface cells. Another technique is the building block model which was applied to deposition simulation [118]. This stochastic technique successively adds material cells with a probability proportional to the local deposition rate.

A further method, developed at the Institute for Microelectronics at the Vienna University of Technology, is based on the so-called structuring element algorithm [93, 122, 123], which is based on the Huygens–Fresnel principle and can be applied to deposition as well as etching processes. Appropriate structuring elements are used to construct the surface front after a certain time step. In the case of isotropic deposition, these structuring elements are spheres with a radius equal to the time step multiplied by the deposition rate. The states of all cells covered by these structuring elements are changed to obtain the profile after each time step. By nature the discrete cell representation of the geometry leads to an unrealistic stepped contour. As a consequence, the determination of the surface orientation requires a costly averaging procedure [123].

Continuous cell values are able to describe the surface position more accurately. Values between 0 and 1 describe the filled [29] or removed [54, 106, 139] fraction of the cell volume. The given surface rates and the conservation of mass result in cellular automata-like update rules for the values of the surface cells. The surface can be extracted using the equi-volume rate model (EVRM) [29], which also allows a more accurate computation of surface normals than the discrete approaches.

Contrary to segment-based methods the cell-based techniques are very robust and insusceptible to topographic changes. Due to the quite simple update rules the implementation is very easy for two- and three-dimensional cases.

3.1.3 The Level Set Method

Another technique to describe moving boundaries over time is the LS method [86]. The surface $\mathcal{S}$ is described as the zero LS of a continuous function $\Phi(\vec{x},t)$ defined on the entire simulation domain

$$\mathcal{S}(t) = \{\vec{x} : \Phi(\vec{x},t) = 0\}. \tag{3.1}$$

Such a function can be obtained by a signed distance transform [53, 77]

$$\Phi(\vec{x}, t=0) := \begin{cases} -\min\limits_{\vec{x}' \in \mathcal{S}(t=0)} \|\vec{x} - \vec{x}'\| & \text{if } \vec{x} \in \mathcal{M}(t=0), \\ +\min\limits_{\vec{x}' \in \mathcal{S}(t=0)} \|\vec{x} - \vec{x}'\| & \text{else.} \end{cases} \tag{3.2}$$

Using the implicit representation of the surface (3.1) the time evolution of the surface driven by a scalar velocity field $V(\vec{x})$ can be described by the LS equation

$$\frac{\partial \Phi}{\partial t} + V(\vec{x}) \|\nabla \Phi\| = 0. \tag{3.3}$$

The velocity field can be obtained from the velocities given on the surface by extrapolation, as will be described later in Section 3.2.4.

The LS method is able to describe the position of the surface with sub-grid accuracy. Furthermore, this method allows an accurate calculation of geometric variables, such as the surface normal or the curvature. Similarly to cell-based methods, the implicit description of the surface position allows handling of topographic changes without special consideration.

After successful demonstrations for the applicability of the LS method to topography simulation [4, 5, 6, 108], this technique has become the most popular technique to track a surface over time, especially in three dimensions. The LS method is used by many academic groups for two-dimensional [8, 112, 113] or three-dimensional topography simulation [46, 65, 96, 98]. The topography simulators earlier developed at the Institute for Microelectronics at the Vienna University of Technology, as ELSA [40] and Topo3D [111], also use the LS method for surface evolution. Furthermore, the newer commercial topography simulators such as the two- and three-dimensional Victory process simulator by Silvaco [43, 126], the two-dimensional Sentaurus Topography simulator by Synopsys [127], or PLENTE by Process-Evolution [16, 17] are also based on the LS method.

3.2 Solving the Level Set Equation

In this section numerical schemes for solving the LS equation are discussed. Since the LS equation belongs to the class of Hamilton-Jacobi equations

$$\frac{\partial \Phi}{\partial t} + H(\vec{x}, \nabla\Phi, t) = 0 \quad \text{for } H(\vec{x}, \nabla\Phi, t) = V(\vec{x}) \|\nabla \Phi\| \tag{3.4}$$

with H denoting the Hamiltonian, many numerical schemes developed for the solution of Hamilton-Jacobi equations can also be applied to the LS equation.

Generally, the solution of a partial differential equation requires a discretization in time as well as in space. In case of the LS equation the simulation domain is usually discretized using regular grids which allow the application of simple finite difference schemes. In the following, Δx denotes the grid spacing. Individual grid points are identified by an index vector $\vec{p} = (p_1, \ldots, p_D) \in \mathcal{G} \subseteq \mathbb{Z}^D$, where $\mathcal{G}$ denotes the set of index vectors of all grid points of the discretized simulation domain.

For regular grids the finite difference method is the most suitable method to solve the LS equation. As the name suggests, it approximates derivates by finite differences. The discretized version of (3.4) using the first-order forward Euler method [92] writes as

$$\Phi^{(t+\Delta t)}(\vec{p}) = \Phi^{(t)}(\vec{p}) - \Delta t \cdot \hat{H}(\vec{p}, \Phi^{(t)}, V(\vec{p})). \tag{3.5}$$

Here $\hat{H}$ is an appropriate finite difference approximation to the Hamiltonian H of the LS equation evaluated at grid point $\vec{p}$. $\hat{H}$ depends on the discretized values of the LS function $\Phi^{(t)}$ at time t and also on the current value of the velocity field $V(\vec{p})$ at grid point $\vec{p}$.

In the following discussion different approximations for the Hamiltonian are presented, which result in different time integration schemes. Since these approximations are based on finite differences, it is useful to introduce a notation for finite differences. The central difference operator is represented by D_i^0

$$D_i^0\Phi(\vec{p}) := \frac{\Phi(\vec{p} + \vec{e}_i) - \Phi(\vec{p} - \vec{e}_i)}{2\Delta x}, \tag{3.6}$$

where $\vec{e}_i$ is the unit vector along the i-th grid direction. Furthermore, the forward difference D_i^+ and the backward difference D_i^- operator are defined as

$$D_i^\pm\Phi(\vec{p}) := \pm\frac{\Phi(\vec{p} \pm \vec{e}_i) - \Phi(\vec{p})}{\Delta x}. \tag{3.7}$$

3.2.1 Upwind Scheme

Based on the Engquist-Osher scheme [28], Osher and Sethian proposed a finite difference upwind scheme to solve the LS equation over time [86, 110]. The scheme uses the following approximation for the Hamiltonian

$$\hat{H}(\vec{p}, \Phi, V(\vec{p})) := \max(V(\vec{p}), 0)\,\nabla^+\Phi(\vec{p}) + \min(V(\vec{p}), 0)\,\nabla^-\Phi(\vec{p}). \tag{3.8}$$

Depending on the sign of the surface velocity V this upwind scheme selects an appropriate one-sided approximation for $\|\nabla\Phi\|$

$$\nabla^\pm\Phi(\vec{p}) := \sqrt{\sum_{i=1}^{D} \left(\max\left(\pm\partial_i^-\Phi(\vec{p}), 0\right)\right)^2 + \left(\min\left(\pm\partial_i^+\Phi(\vec{p}), 0\right)\right)^2} \tag{3.9}$$

to guarantee a stable solution. $\partial_i^{\pm}\Phi$ are one-sided approximations to the first derivatives with respect to the i-th direction

$$\partial_i^{\pm}\Phi(\vec{p}) := D_i^{\pm}\Phi(\vec{p}) \underbrace{\mp \frac{\Delta x}{2}\zeta(D_i^{\pm}D_i^{\pm}\Phi(\vec{p}), D_i^{+}D_i^{-}\Phi(\vec{p}))}_{\text{second order term}}. \tag{3.10}$$

ζ is a switch function and is defined as

$$\zeta(a,b) := \begin{cases} a & \text{if } (0 \leq a \leq b) \vee (0 \geq a \geq b), \\ b & \text{if } (0 \leq b < a) \vee (0 \geq b > a), \\ 0 & \text{else.} \end{cases} \tag{3.11}$$

Depending on whether the second order term in (3.10) is included or not the previous equations describe the second or the first order space scheme as given in [110]. The advantage of these schemes is that they can be straightforwardly implemented without special knowledge of the surface velocity function V. However, stability is only guaranteed for convex speed laws. That means the scheme is stable for convex Hamiltonians with respect to the first order derivatives of the LS function $q_i := \frac{\partial \Phi}{\partial x_i}$. This is the case, if the matrix $\left(\frac{\partial^2 H}{\partial q_i \partial q_j}\right)$ is positive-semidefinite [110].

Unfortunately non-convex Hamiltonians occur frequently in topography simulations. In the general case the surface velocities depend on the surface normal (see for example (2.24) or (2.36)). As shown later in Section 3.3.1 the surface normal can be expressed using the first order derivatives of the LS function. Hence, the convexity of the Hamiltonian is actually given by the model specific surface velocity function.

Non-convex Hamiltonians occur often, if yield functions are involved, which depend on the incident angle as described by (2.21). Models for anisotropic wet etching processes (see Section 2.3.3) can also lead to non-convex Hamiltonians.

3.2.2 Lax-Friedrichs Scheme

To solve the LS equation for non-convex Hamiltonians the Lax-Friedrichs scheme can be applied [87, 110]. Some artificial dissipation is introduced to obtain stability. The approximation to the Hamiltonian for this scheme is written as

$$\hat{H}(\vec{p}, \Phi, V(\vec{p})) := V(\vec{p})\nabla^0\Phi(\vec{p}) - \sum_{i=1}^{D} \alpha_i \cdot \frac{\partial_i^{+}\Phi(\vec{p}) - \partial_i^{-}\Phi(\vec{p})}{2} \tag{3.12}$$

using the central gradient approximation

$$\nabla^0\Phi(\vec{p}) := \sqrt{\sum_{i=1}^{D}\left(\frac{\partial_i^{-}\Phi(\vec{p}) + \partial_i^{+}\Phi(\vec{p})}{2}\right)^2}. \tag{3.13}$$

$\partial_i^{\pm}\Phi$ are again the one-sided approximations defined in (3.10). Again, depending on whether the second order term in (3.10) is included or not, the scheme is second or first order accurate in space, respectively [110].

The constants α_i are dissipation coefficients which should fulfill

$$\alpha_i \geq \max_{\vec{x}} \left| \frac{\partial H}{\partial q_i} \right| \quad \text{with } q_i := \frac{\partial \Phi}{\partial x_i} \tag{3.14}$$

in order to guarantee stability [85, 110].

The choice of these dissipation coefficients is the main problem for the applicability of this scheme. Larger values lead to unnecessary smoothing of the surface, while smaller values cause numerical instabilities. The analytical evaluation of the right-hand side in (3.14) to find the optimal values for the constants α_i is usually not possible for general surface velocities. However, often the optimal values cannot be straightforwardly evaluated, because the velocity field may depend on the particle transport which further depends on the surface geometry. Hence, the likely best way to find the optimal values is to test different values and analyze the results [98].

3.2.3 Stability

In time integration schemes, such as the first-order forward Euler method, information from a grid point to a neighbor grid point can at most propagate with the velocity given by the ratio of the grid spacing and the time increment, $\frac{\Delta x}{\Delta t}$. In order to calculate the movement of a surface, its maximum speed must be smaller than this critical value

$$\max_{\vec{p} \in \mathcal{G}} |V(\vec{p})| \leq \frac{\Delta x}{\Delta t}. \tag{3.15}$$

This condition needs to be fulfilled in order to guarantee a stable time integration and is known as the Courant-Friedrichs-Lewy (CFL) condition [24].

For each integration step the time increment must be adapted to fulfill the CFL condition. In practice, a positive constant $C_{\text{CFL}} \in]0, 1]$, the so-called CFL number, is defined and an appropriate time increment is chosen according to

$$\Delta t = C_{\text{CFL}} \cdot \frac{\Delta x}{\max\limits_{\vec{p} \in \mathcal{G}} |V(\vec{p})|}. \tag{3.16}$$

3.2.4 Surface Velocity Extension

In topography simulations the velocity field V has no physical meaning, since the surface velocity can only be determined at points on the surface, where the surface velocity corresponds to the local deposition or etch rate. The LS method, however,

requires a velocity field, for which the values must be known for at least all grid points which are updated in time. Hence, to enable the solution of the LS equation a technique is required which extrapolates the surface velocities from the surface to all those grid points.

Assuming that $V(\vec{x})$ is available for all points $\vec{x} \in \mathcal{S}(t)$, different techniques have been developed to obtain a surface velocity field which allows a stable time integration of the LS equation. In [73], it was proposed to define the surface velocity field as

$$V(\vec{x}) := V(\vec{x}_{\mathrm{cp}}(\vec{x})) \quad \text{with} \quad \vec{x}_{\mathrm{cp}}(\vec{x}) := \arg\min_{\vec{x}' \in \mathcal{S}} \|\vec{x}' - \vec{x}\|. \tag{3.17}$$

Hence, the surface velocity at a certain point $\vec{x}$ is set to the velocity of its closest surface point $\vec{x}_{\mathrm{cp}}(\vec{x}) \in \mathcal{S}$. Since the velocity field needs to be known for all grid points, all their corresponding closest surface points must be determined.

A less costly technique to extrapolate the surface velocities based on an alternative formulation of the problem [7] is

$$\|\nabla \Phi_{\mathrm{temp}}\| = 1, \tag{3.18}$$

$$\Phi_{\mathrm{temp}}(\vec{x}) = 0 \Leftrightarrow \Phi(\vec{x}) = 0, \tag{3.19}$$

$$\nabla V \cdot \nabla \Phi_{\mathrm{temp}} = 0. \tag{3.20}$$

Here, the first two equations define a signed distance function Φ_{temp} with the same zero LS as Φ. The solution of this boundary value problem fulfills (3.17). The system of equations can be efficiently solved using the fast marching method [109]. With this finite difference scheme the surface velocity field can be obtained for all grid points with an optimal linear complexity.

3.3 Approximations to Geometric Variables

One strength of the LS method is the simple calculation of geometric variables, such as the surface normal or the curvature, directly from the implicit surface representation given by the LS function.

3.3.1 Surface Normal

Generally, the normal vector at point $\vec{x}$ on the LS of a smooth function is given by

$$\vec{n}(\vec{x}) = \frac{\nabla \Phi}{\|\nabla \Phi\|}. \tag{3.21}$$

At grid points $\vec{p} \in \mathcal{G}$ the i-th component of the normal vector can be approximated by

$$n_i(\vec{p}) \approx \frac{D_i^0 \Phi(\vec{p})}{\sqrt{\sum_{j=1}^{D} \left(D_j^0 \Phi(\vec{p}) \right)^2}}. \tag{3.22}$$

Here D_i^0 is the central difference operator as defined in (3.6). The normal vector for a grid point close to the surface $\mathcal{S}$ represented by the zero LS is also a good approximation for the normal on the surface for the closest surface point. The closest surface point $\vec{x}_{\mathrm{cp}}(\vec{p}) \in \mathcal{S}$ of a nearby grid point $\vec{p}$ can be approximated by [135]

$$\vec{x}_{\mathrm{cp}}(\vec{p}) \approx \vec{p} - \vec{n}(\vec{p}) \cdot \frac{\Phi(\vec{p})}{\|\nabla \Phi\|}, \tag{3.23}$$

if the grid point indices $\vec{p}$ are equal to the grid point coordinates. Here the last factor corresponds to the approximated signed distance to the surface. For the denominator the same approximation is used as in (3.22).

3.3.2 Curvature

Solving the LS equation with a velocity field that is proportional to the curvature can be used for smoothing as will be demonstrated in Section 4.5. For an arbitrary point $\vec{x}$ the (mean) curvature of the corresponding LS is defined as

$$\kappa(\vec{x}) = \nabla \vec{n}(\vec{x}) = \nabla \cdot \frac{\nabla \Phi}{\|\nabla \Phi\|}. \tag{3.24}$$

At grid points $\vec{p} \in \mathcal{G}$ the mean curvature can be approximated by [110]

$$\kappa(\vec{p}) = \frac{\sum_{i \neq j} \left(D_i^0 \Phi(\vec{p}) \right)^2 \cdot D_j^+ D_j^- \Phi(\vec{p}) - D_i^0 \Phi(\vec{p}) \cdot D_j^0 \Phi(\vec{p}) \cdot D_i^0 D_j^0 \Phi(\vec{p})}{\left(\sum_{i=1}^{D} \left(D_i^0 \Phi(\vec{p}) \right)^2 \right)^{\frac{3}{2}}}. \tag{3.25}$$

3.4 Acceleration Techniques

The LS method as initially proposed in [86] uses a LS function which is defined on the entire simulation domain. Consequently, the memory requirements for the discretization of the LS function scale with the domain size. The scaling law of the computation time for the time update scheme is similar. However, the optimal complexity for storing and evolving a surface over time should linearly depend on the surface area (measured in grid spacings). Different approaches have been developed in the past to make the LS method more efficient.

3.4.1 The Narrow Band Method

To obtain a linear scaling law for the computation time the narrow band method was introduced [3]. This approach makes use of the fact that only the LS values of grid points close to the surface have an influence on the zero level set and thus on the actual position of the surface. By updating only a subset of grid points in time, namely only those within a band around the surface with a typical width of a couple of grid spacings, the computation time can be drastically reduced. Since the number of these so-called active grid points within the band is approximately the surface area times the narrow band width, an optimal linear scaling for the time integration is obtained. From time to time, the narrow band needs to be re-initialized, if the surface approaches its boundary.

3.4.2 The Sparse Field Method

A further development of the narrow band method is the sparse field method [135], which further reduces the computational effort by considering just a single layer of active grid points for time integration.

The method classifies all grid points $\vec{p} \in \mathcal{G}$ depending on their LS value as follows

$$\mathcal{L}_i := \begin{cases} \{\vec{p} \in \mathcal{G} : i - \frac{1}{2} \leq \Phi(\vec{p}) < i + \frac{1}{2}\} & \text{if } i < 0, \\ \{\vec{p} \in \mathcal{G} : -\frac{1}{2} \leq \Phi(\vec{p}) \leq \frac{1}{2}\} & \text{if } i = 0, \\ \{\vec{p} \in \mathcal{G} : i - \frac{1}{2} < \Phi(\vec{p}) \leq i + \frac{1}{2}\} & \text{if } i > 0. \end{cases} \tag{3.26}$$

The sparse field LS method assumes that from any two neighboring grid points with different signed LS values at least one belongs to $\mathcal{L}_0$

$$\forall \vec{p} \in \mathcal{G}, \vec{p}' \in \eta(\vec{p}) : \operatorname{sgn}(\Phi(\vec{p})) \neq \operatorname{sgn}(\Phi(\vec{p}')) \Rightarrow \{\vec{p}, \vec{p}'\} \cap \mathcal{L}_0 \neq \emptyset. \tag{3.27}$$

Here $\eta(\vec{p})$ denotes the set of the closest neighbor grid points of $\vec{p}$. In an infinite regular grid each grid point has $2D$ such neighbors, if D is the number of dimensions. The signum function is assumed to be two-valued; $\operatorname{sgn}(0)$ can be either 1 or -1. For numerical float data types the sign bit can be used to evaluate the sign. (3.27) is equivalent to the statement that the grid points with positive and negative signed LS values are separated by those belonging to $\mathcal{L}_0$.

A further requirement of the sparse field LS method is that the layers $\mathcal{L}_i$ fulfill

$$\mathcal{L}_i = \left\{ \vec{p} \in \mathcal{G} : \operatorname{sgn}(\Phi(\vec{p})) \cdot \min_{\vec{p}' \in \mathcal{L}_0} \|\vec{p} - \vec{p}'\|_1 = i \right\}, \tag{3.28}$$

where $\|\cdot\|_1$ denotes the Manhattan norm. Once the grid points fulfill (3.26) to (3.28) the sparse field LS method can be applied. The method itself maintains these properties over time. A proper technique to initialize the LS values appropriately is described later in Section 4.1.1.

The idea of the sparse field method is now to consider only active grid points, namely those belonging to the most inner layer $\mathcal{L}_0$, for time integration. However, the previously introduced numerical schemes also need the LS values of neighbor grid points, thus those of neighboring layers. For example, if second order derivatives are needed the LS values of grid points in layers $\mathcal{L}_{\pm1}$ and $\mathcal{L}_{\pm2}$ are required. After each time step, the LS values of non-active grid points must also be updated. Furthermore, the sets of grid points belonging to the individual layers may change and must be redetermined. The different sets of grid points $\mathcal{L}_i$ are originally represented as a list of pointers to the corresponding memory representations. The basic procedure of the sparse field method is presented in the following steps:

1. Update the LS values of all active grid points $\vec{p} \in \mathcal{L}_0^{(t)}$ in time using an appropriate finite difference scheme as described in the previous sections.
2. If there are any two neighboring grid points $\vec{p}, \vec{p}\,' \in \mathcal{L}_0$ for which $\Phi(\vec{p}) < -\frac{1}{2}$ and $\Phi(\vec{p}\,') > \frac{1}{2}$ the absolute LS values of both points are reduced to $\Phi(\vec{p}) = -\frac{1}{2}$ and $\Phi(\vec{p}\,') = \frac{1}{2}$, respectively.
3. In the order $i = \pm1, \pm2, \ldots$ the LS values of all grid points in the i-th layer $\vec{p} \in \mathcal{L}_i^{(t)}$ are updated using

$$\Phi^{(t+\Delta t)}(\vec{p}) = \begin{cases} \min\limits_{\vec{p}\,' \in \eta(\vec{p}) \cap \mathcal{L}_{i-1}^{(t)}} \Phi^{(t+\Delta t)}(\vec{p}\,') + 1 & \text{if } i > 0 \\ \max\limits_{\vec{p}\,' \in \eta(\vec{p}) \cap \mathcal{L}_{i+1}^{(t)}} \Phi^{(t+\Delta t)}(\vec{p}\,') - 1 & \text{if } i < 0 \end{cases}. \tag{3.29}$$

4. Finally, the layers $\mathcal{L}_i^{(t+\Delta t)}$ for the next time step are determined using (3.26). Alternatively, due to the nature of the update scheme the determination of the layers using (3.28) is equivalent.

As previously mentioned, depending on the finite difference approximations used in the numerical update scheme a certain neighborhood around the active layer $\mathcal{L}_0$ is required. For example, if second order space schemes are used the availability of the LS values of all grid points in the layers $\mathcal{L}_0, \mathcal{L}_{\pm1}, \mathcal{L}_{\pm2}$ is sufficient. Consequently, the update rule (3.29) can be limited to grid points which are contained by these layers in the next time step.

Item 2 does not exist in the original description of the sparse field method [135]. It has been proposed in [A18] to improve the robustness of the algorithm. If arbitrary velocity fields are allowed, cases may occur which contradict the assumption (3.27). As depicted in Figure 3.1 the update of the opposite signed LS values of two neighboring active grid points can lead to a pair of neighboring grid points, none of which belongs to $\mathcal{L}_0$.

Another problem which may occur was already mentioned in the original paper [135]. The update scheme potentially leads to dense sets of active grid points, which do not have an opposite signed neighbor grid point. Figure 3.2 shows two examples,

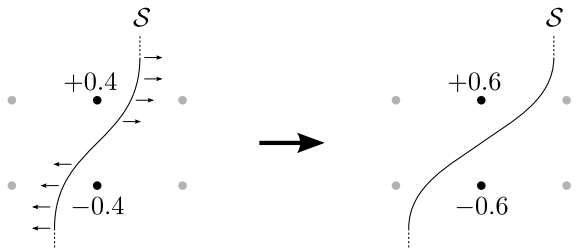


Figure 3.1: A problematic case which may occur and which needs special consideration. For the update of the neighboring pair of active grid points (black) with LS values ± 0.4 opposite signed surface velocities (arrows) are used. This leads to a neighboring pair of non-active grid points with opposite signed LS values.

where so-called inefficient sets of active grid points are produced. They often appear in regions, where the surface converges, and do not necessarily need to be close to the surface afterwards, as illustrated by the second example. Hence, these points do not contribute to the description of the surface, and updating them is not necessary. The consideration of these non-relevant grid points makes the expansion of the surface velocity field more complicated and computationally more intensive. Even worse, dense sets of such points could be produced, essentially increasing the memory consumption and the calculation time during time integration. To avoid the accumulation of such active grid points a pruning procedure was proposed to keep $\mathcal{L}_0$ an efficient set of active grid points, so that each active grid point has an opposite signed neighbor point:

$$\forall \vec{p} \in \mathcal{L}_0 : \exists \vec{p}' \in \eta(\vec{p}) : \operatorname{sgn} \Phi(\vec{p}) \neq \operatorname{sgn} \Phi(\vec{p}'). \tag{3.30}$$

The pruning procedure removes active grid points from $\mathcal{L}_0$, which do not fulfill this criterion and finally updates all layers $\mathcal{L}_i$ and the corresponding LS values using (3.28) and (3.29), respectively. In Section 4.3 a robust algorithmic realization of the sparse field method including the pruning procedure will be presented.

For the sparse field method the CFL condition (3.15) can be written as

$$\max_{\vec{p} \in \mathcal{L}_0^{(t)}} \left| \Phi^{(t+\Delta t)}(\vec{p}) - \Phi^{(t)}(\vec{p}) \right| \leq 1, \tag{3.31}$$

which ensures that the surface advances at most one grid spacing per time step. For each step the time increment Δt is chosen in such a way that

$$\max_{\vec{p} \in \mathcal{L}_0^{(t)}} \left| \Phi^{(t+\Delta t)}(\vec{p}) - \Phi^{(t)}(\vec{p}) \right| = C_{\mathrm{CFL}} \tag{3.32}$$

is fulfilled, where C_{CFL} again denotes the predefined CFL number. A good choice for this positive constant is a value smaller than or equal to 0.5, which guarantees that only grid points of the active layer $\mathcal{L}_0^{(t)}$ change the sign of their LS value within a time step. This follows from the fact that the absolute LS values of grid points $\vec{p} \notin \mathcal{L}_0^{(t)}$

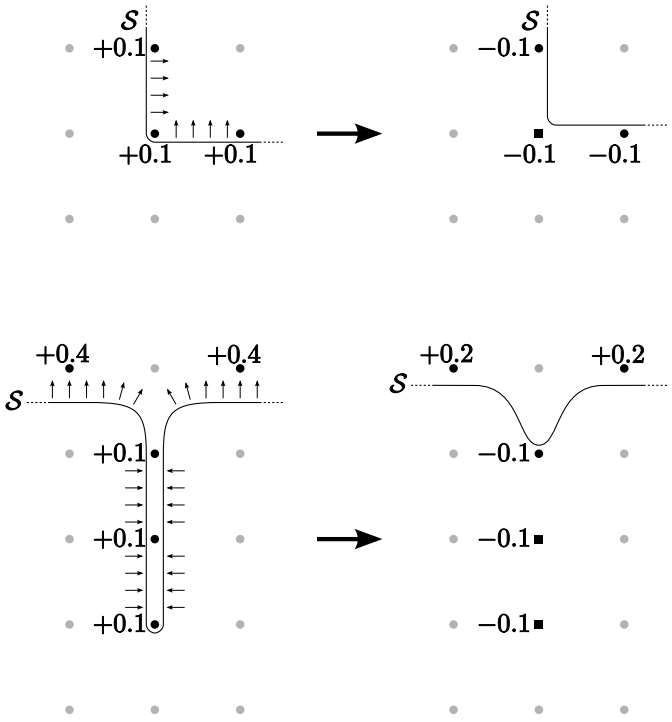


Figure 3.2: Two examples, where the sparse field method produces inefficient sets of active (black) grid points. The surface $\mathcal{S}$ moves with a uniform positive surface velocity (arrows). After a time step some active grid points (squares) do not have a neighbor with opposite signed LS value.

are, according to (3.26), larger than 0.5 while the maximum change of the LS value is limited by C_{CFL}, as stated in (3.32). The time step can be directly calculated by inserting (3.5) in (3.32)

$$\Delta t = \frac{C_{\mathrm{CFL}}}{\max\limits_{\vec{p}\in\mathcal{L}_0^{(t)}} \left|\hat{H}(\vec{p}, \Phi^{(t)}, V(\vec{p}))\right|}. \tag{3.33}$$

The sparse field method saves more computation time than the narrow band method for several reasons. First, only a minimal set of active grid points is involved in the time integration procedure. Furthermore, the time consuming surface velocity extension can be avoided. This extension is necessary for topography simulations, since the velocities are only defined on the surface and the LS method requires a velocity field [7]. Finally, the sparse field method does not require regular re-initializations, unlike the narrow band method [3]. The sparse field method was first applied in the field of topography simulation in [94].

3.5 Level Set Data Structures

The presented acceleration techniques do not need the LS values of all points in the grid. Hence, it is sufficient to store only the LS values of all defined grid points. The set of defined grid points is the union of all active grid points plus those of neighboring grid points which are needed to construct the finite differences as required by the update scheme. In the following, an overview of efficient data structures for storing the discretized LS function is presented.

3.5.1 Trees

Quad- and octtrees for two and three dimensions, respectively, have been successfully applied to the LS method [46, 71, 81, 121]. The LS values of defined grid points are stored within leaf nodes and can be accessed in logarithmic time. By using an adaptive grid with increasing resolution close to the surface, an optimal linear scaling law with surface area can be obtained. At the surface the same maximum resolution is generally used, because the solution of the LS equation with the finite difference method is more complex and less accurate on non-uniform grids.

Despite the optimal scaling law, the internal pointer structure of trees leads to a significant memory overhead. Furthermore, the memory layout of trees is usually not very convenient for fast serial processing.

3.5.2 Run-Length Encoding

Instead of using an adaptive grid to represent the entire simulation domain, it is better to use a regular grid with an appropriate resolution and store only the LS values of defined grid points needed for the time integration scheme.

Run-length encoding along a certain grid direction is an efficient technique for this purpose [45]. In the following, run-length encoding is explained for the x_1-direction. However, it can be applied to all other grid directions analogously. Each line of grid points parallel to the x_1-direction is separately run-length encoded. Consecutive undefined grid points, for which the LS values do not need to be stored, are combined in runs. It is advantageous to use two different run codes, depending on whether the corresponding run contains only grid points where the LS function is positive or negative, respectively. In this way the sign of the LS function is available for all grid points. This information is useful, since it reveals on which side of the surface a grid point is located.

To demonstrate, run-length encoding is applied to the two-dimensional example shown in Figure 3.3. There, a triangle is described by the LS. The LS function is resolved on a grid with extensions 12×12. Figure 3.4 shows the corresponding run-length encoded

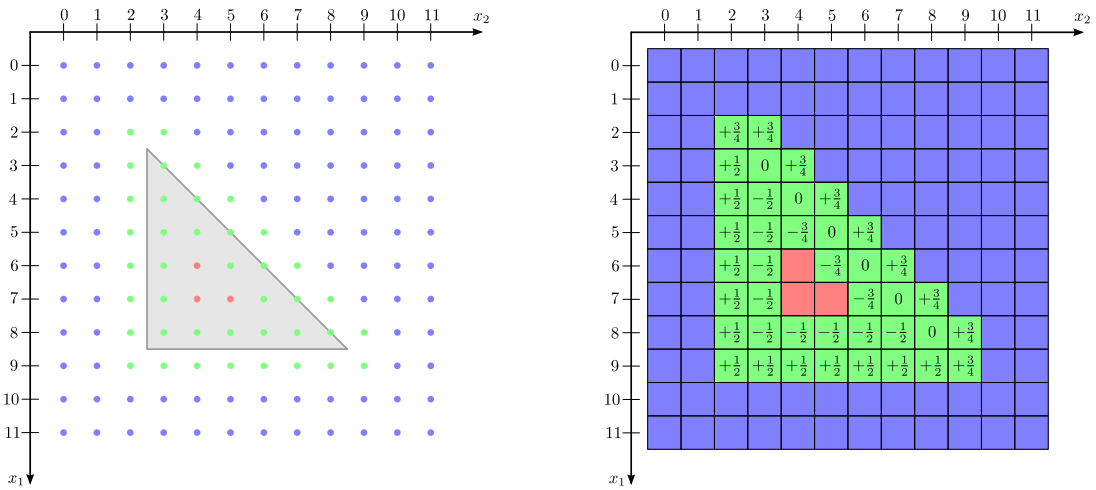


Figure 3.3: A LS function representing a triangle is resolved on a 12×12 grid. Only the LS of defined grid points (green) must be stored. Memory can be saved by not storing the LS values of all other positive (blue) and negative (red) undefined grid points. The LS values of all defined grid points are given on the right-hand side.

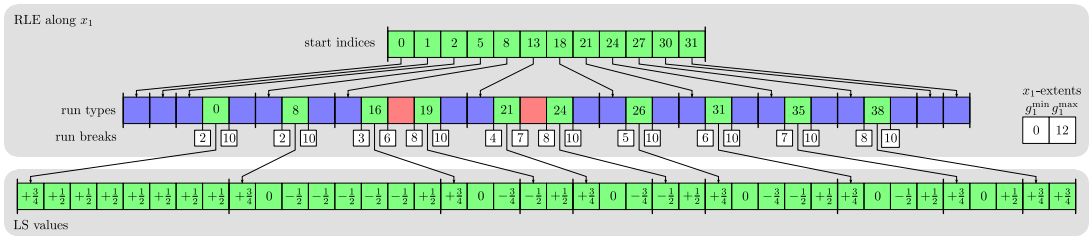


Figure 3.4: The RLE data structure for the example given in Figure 3.3. For each of the 12 grid lines along the x_1-direction an index is stored in the start indices array to the corresponding run type sequence. Three different types of run codes can be distinguished: Undefined runs which are either positive (blue) or negative (red), and defined runs (green) whose run codes give the corresponding start indices in the LS values array. Another array stores the run breaks from which the start and end indices of a run can be obtained. The size of the run breaks array is always equal to the difference of the sizes of the run types array and the start indices array.

(RLE) data structure which consists of 4 arrays in order to store only the LS values of defined grid points near the boundary. To address array entries, zero based indexing is used.

The LS values of all defined grid points are stored in lexicographical order in the LS values array. Sequences of positive undefined, negative undefined, or defined grid points are represented by integral run codes stored within the run types array. In case of a defined sequence the run code gives the corresponding start index in the LS values array. Grid indices at which new runs start are given in the run breaks array.

Direct access to run codes for a specific line of grid points is provided by the start indices array. The size of this array is equal to the number of grid lines parallel to the x_1-direction. In the two-dimensional case the size of this array is equal to the grid extension in the x_2-direction, which is equal to 12 in this example. The index of the first run break for a given grid line can be obtained by subtracting the index of the corresponding entry in the start indices array from the content of the same entry.

The RLE data structure provides fast random access to LS values of defined grid points. The LS value of a certain grid point $\vec{p} = (p_1, \ldots, p_D)$ can be found by accessing the corresponding RLE grid line through the start indices array. The array index of the grid line is calculated as $\sum_{i=2}^{D}(p_i - g_i^{\min}) \prod_{j=2}^{i-1}(g_j^{\max} - g_j^{\min} + 1)$, where $\vec{g}^{\min}$ and $\vec{g}^{\max}$ denote the minimum and maximum index vectors of the grid. A binary search of p_1 within the corresponding run breaks is necessary to find the correct run code. If the run is defined, the index of the LS value of $\vec{p}$ can be calculated using the run code, the first grid index of the run obtained from the run breaks array or the grid extensions, and p_1.

The complexity of accessing the LS value of a specific grid point is mainly given by the binary search, which scales logarithmically with the number of runs within a grid line. In the worst case, if all defined grid points belong to the same grid line and are separated by runs of undefined grid points (undefined runs), the complexity is of order $\mathcal{O}(\log N_{\mathrm{D}})$, where N_{D} denotes the number of defined grid points. However, in practice a much better performance can be expected.

The memory requirements of the RLE data structure scale according to $\mathcal{O}(N_{\mathrm{D}} + \prod_{i=2}^{D}(g_i^{\max} - g_i^{\min} + 1))$. The lexicographical order of LS values is very advantageous, because it allows fast serial processing. However, in contrast to tree data structures, since the data structure is based on arrays, it is not possible to add or remove grid points efficiently, except at the end. If the set of defined grid points changes, the entire data structure must be reset.

3.5.3 Dynamic Tubular Grid

Another data structure for storing only the LS values of defined grid points is the dynamic tubular grid (DTG) [84]. The DTG is hierarchically organized over the dimensions and is able to store a certain subset of grid points together with some data. The DTG can be used to store sparse data on grids with an arbitrary number of dimensions. The basic structure of a DTG is shown in Figure 3.5.

The basic idea of the DTG is to apply subsequent orthogonal projections along all grid directions on the set of defined grid points. First, a projection along the x_1-direction leads to a set of defined grid points in a lower-dimensional subspace, which is stored using a DTG with lower dimensionality. Hence, the DTG is recursively defined. For each projected defined grid point an index is stored in this lower-dimensional DTG. This index gives access to all the defined grid points which are projected to the same

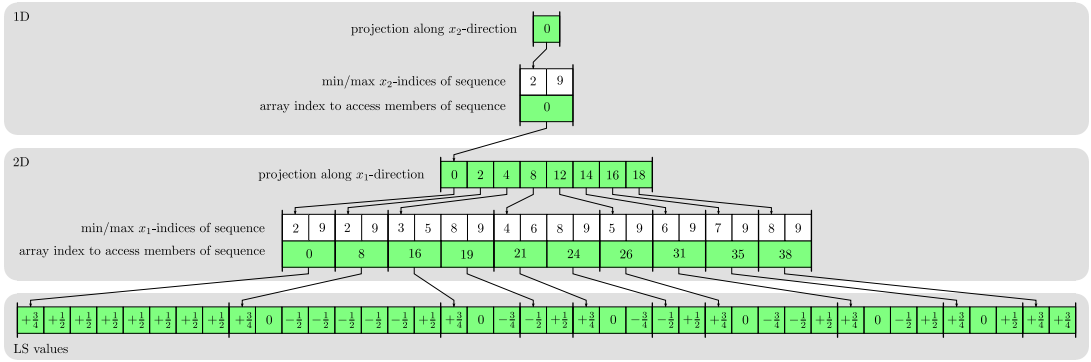


Figure 3.5: The defined grid points and their associated LS values of the example shown in Figure 3.3 as stored in a DTG. The hierarchical structure is obtained by subsequent projection of grid points to a lower dimension, and combining connected sequences of indices along a grid line. The minimum and maximum indices of such a sequence are stored together with an array index which provides access to its members. Since the topmost array always contains just a single entry with value 0, it can be omitted. This array is not included in the original definition of the DTG and is only shown to emphasize the hierarchical structure.

lower-dimensional grid point. Sequences of defined grid points along the projection direction are stored in a compact way using the minimum and maximum indices and an index giving access to the individual members of the sequence.

The memory requirements of the DTG scale linearly with the number of defined grid points, N_D. Random access to the data of grid points requires a binary search within the array holding the minimum and maximum indices of sequences along each grid direction. Similarly to the RLE data structure, the binary searches lead to a worst case logarithmic complexity $\mathcal{O}(\log N_\mathrm{D})$. In practice, since the data structure is very cache coherent, random access is almost as fast as for a full grid.

Sequential access can be realized in constant time using iterators moving over the data structure in lexicographical order as described in [84]. Neighbor access to grid points, which is essential in calculating finite differences, can also be performed in constant time, if a stencil of iterators is moved simultaneously over the data structure.

The DTG allows storage of defined grid points on an infinite domain. The indices of points can take arbitrary values and do not need to be within a certain range, as is the case for a full grid representation. (The index values are only limited by the minimum and maximum possible representable values of the used integral data type.) For a full grid to mimic an infinite grid, grid extensions must be adapted, when the LS reaches the full grid boundary.

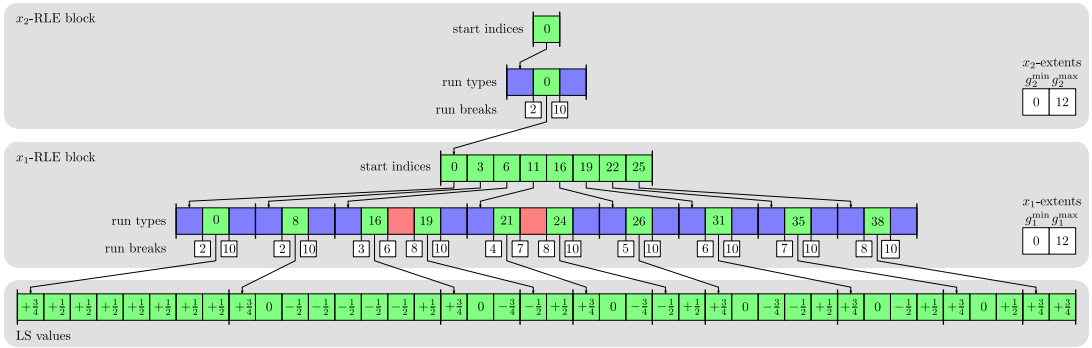


Figure 3.6: The H-RLE data structure applies run-length encoding hierarchically over all grid dimensions in order to obtain an efficient LS data structure which combines the advantages of run-length encoding and the hierarchically organized DTG.

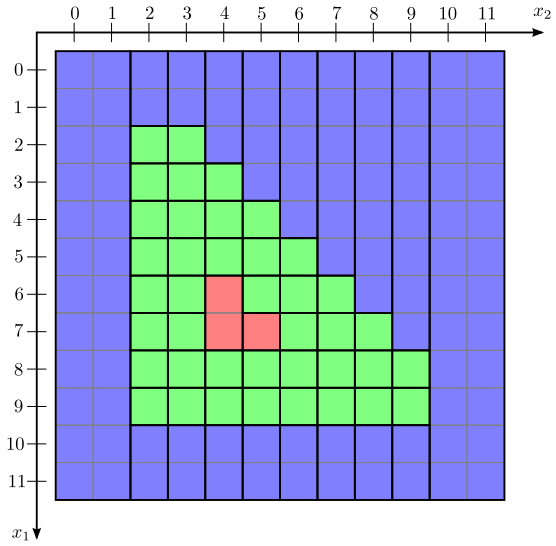


Figure 3.7: The segmentation of the grid (black thick lines) which is implicitly defined by the H-RLE data structure. Each segment represents either a run of positive (blue) or negative (red) undefined grid points or a single defined grid point (green).

3.5.4 Hierarchical Run-Length Encoding

The RLE data structure is also able to store the sign of the LS function for all undefined grid points, while the DTG has ideal linear scaling memory requirements and is adaptive in all grid directions. A data structure that combines the advantages of the RLE and DTG data structures is the hierarchical run-length encoded (H-RLE) data structure [44].

As shown in Figure 3.6 the H-RLE data structure is hierarchically organized, similar to the DTG. However, instead of storing sequences of defined grid points, which are

projected to the same grid point in the lower dimensional space, run-length encoding is applied. Hence, the H-RLE data structure is able to store the sign of the LS function for all undefined grid points and, in addition, it shows the same characteristics regarding memory consumption and random access as the DTG.

The H-RLE data structure leads to an inherent segmentation of the entire grid. All grid points either belong to an undefined run or are defined grid points. Figure 3.7 shows the corresponding segmentation of the two-dimensional example given in Figure 3.3. The number of segments for the worst case is $N_{\mathrm{D}} \cdot (2D + 1) = \mathcal{O}(N_{\mathrm{D}})$. However, if many defined grid points are neighbored, which is a common case for the LS method, the number of segments is much smaller.

4 A Fast Level Set Framework

This chapter describes the realization of a fast LS framework based on the sparse field LS method and the H-RLE data structure. The framework is able to describe multiple material regions and also supports Boolean operations which are particularly useful for geometrical operations. Furthermore, by taking advantage of the H-RLE data structure, fast algorithms for a unidirectional visibility test and for void detection are realized. Finally, a parallelization strategy for algorithms on the H-RLE data structure is presented.

4.1 Initialization

Dependent on the number of dimensions, the initial boundary is usually given as an oriented line segmentation or an oriented triangulation, respectively. The orientation of a line segment or a triangle is defined in this work as shown in Figure 4.1. The normal vector always points to the positive region of the LS function. Without loss of generality, grid point coordinates are assumed to be equal to their indices $\vec{p}$. As a consequence, the grid spacing is unity ($\Delta x = 1$).

4.1.1 Closest Point Transformation

As mentioned in Section 3.1.3 the LS function is usually initialized as a signed distance function. In case of the sparse field LS method it is better to use the smallest (signed) Manhattan distance

$$\Phi(\vec{p}) := \pm \min_{\vec{x} \in \mathcal{S}} \|\vec{p} - \vec{x}\|_1, \tag{4.1}$$

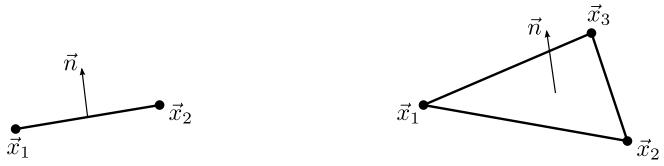


Figure 4.1: The orientation of the normal vector $\vec{n}$ on a line segment and on a triangle as defined in this work with respect to the order of nodes $\vec{x}_i$.

rather than the smallest Euclidean distance, for the initialization. With the latter, the first time step of the sparse field LS method gives wrong results for the position of a (non-axis-parallel) plane moving with constant speed. However, if initialized with the Manhattan distance, trilinear interpolation of the LS function correctly describes the position of the plane after the first time step. The reason for this behavior is the update scheme (3.29) which also corresponds to a Manhattan distance calculation.

It is not necessary to calculate the initial LS values for the entire grid. Only grid points close to the boundary have to be considered. In particular, the sparse field method requires the determination of all active grid points together with their LS values only at the beginning. If the sign of the LS function is known for all other grid points, the additional layers can be determined using (3.28). The required LS values can then be computed using (3.29). The knowledge of the sign for the LS values at all points of the grid is also a prerequisite for the setup of the H-RLE data structure.

The signs of the LS values are unambiguously defined, if the set of all grid points with an opposite signed neighbor is known. This set of grid points clearly separates the positive and the negative grid points from each other. To satisfy all the requirements for the initialization of the sparse field method and the H-RLE data structure, the LS framework determines all grid points for which the LS value is in the range $[-1, 1]$.

This set of grid points contains all the required information, because due to (4.1) all grid points are included, which have an opposite signed neighbor. Before setting up the H-RLE data structure, the LS framework collects all these grid points together with their LS values and stores them in a list. In the following sections an efficient technique is described to obtain this list in linear time.

Since the Manhattan distance is only needed for grid points close to the boundary, the distance computation can be simplified. For these points the Manhattan distance can be approximated by the smallest distance to the boundary along any paraxial direction. In other words, the smallest unsigned distance of a grid point is approximated by

$$|\Phi(\vec{p})| := \min\{\alpha \geq 0 : \exists k : \vec{p} + \alpha \cdot \vec{e}_k \in \mathcal{S} \ \vee \vec{p} - \alpha \cdot \vec{e}_k \in \mathcal{S}\}. \tag{4.2}$$

Here $\vec{e}_k$ denotes the unit vector pointing in the x_k-direction ($1 \leq k \leq D$).

In practice the distance transform is carried out by an iteration over all segments of the discretized boundary representation. For each segment all intersecting grid lines are determined. The number of possible grid lines can be confined using the bounding box of the segment. In the two-dimensional case the test whether a grid line intersects a line segment or not is trivial. In the three-dimensional case the intersection test with a triangle is more difficult and is described in Appendix A. For each intersecting grid line the intersection point $\vec{q} = (q_1, \ldots, q_D)$ is calculated and all grid points $\vec{p}$ on that grid line are considered for which

$$|p_k - q_k| \leq 1 + \varepsilon_1. \tag{4.3}$$

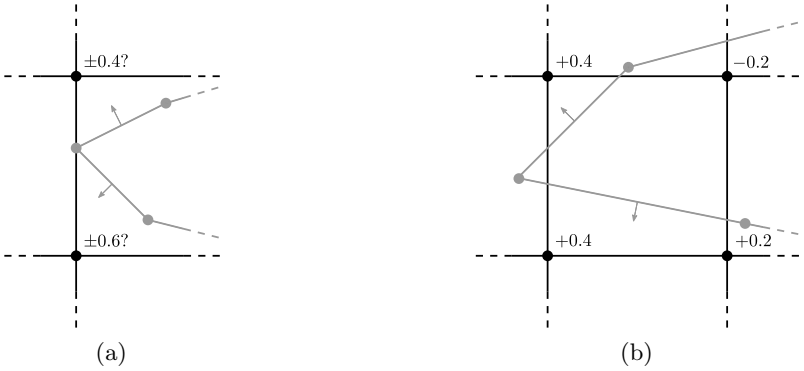


Figure 4.2: (a) Two segments of the surface mesh (gray) meet on a grid line (black). Hence, for both grid points the distance is equal to both segments. As consequence the determination of the signed distance according to (4.4) is ambiguous. (b) The distance transform can produce inefficient sets of active grid points. The bottom left grid point does not have an opposite signed neighbor.

Here it is assumed that the grid line is parallel to the k-th axis direction, hence parallel to $\vec{e}_k$. ε_1 is a small positive constant ($0 < \varepsilon_1 \ll 1$), introduced for numerical reasons, which ensures the calculation of the initial LS values for all grid points directly neighbored to the boundary. Thus, ε_1 must be larger than the maximum numerical error for the calculation of the distance $|p_k - q_k|$. In this work $\varepsilon_1 = 10^{-4}$ is used.

The signed distance of a grid point $\vec{p}$ to the current segment is given by

$$\tau(\vec{p}, k) = \operatorname{sgn}(n_k) \cdot (p_k - q_k)\,. \tag{4.4}$$

The sign of the k-th component of the normal vector $\vec{n}$ can be obtained by considering the orientation of the surface segment with respect to $\vec{e}_k$. In two dimensions the distance is positive, if the vertex $\vec{x}_1$ is on the left-hand side. In case of three dimensions the sign is positive, if the vertices $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$ are seen counterclockwise.

If for each grid point $\vec{p}$ the distance with the smallest absolute value $|p_k - q_k|$ is kept, at least all grid points with LS values in the range $[-1, 1]$ will be properly initialized after iterating over all segments of the boundary mesh. However, this procedure can lead to problems as depicted in Figure 4.2a, where the wrong sign could be assigned to the LS value of both grid points, since both segments are equally distanced. To resolve this ambiguity without additional consideration of neighbor segments, another distance is measured using

$$\tau'(\vec{p}, k) = \operatorname{sgn}(p_k - q_k) \cdot (p_k - q_k - \varepsilon_2 \cdot t_k) \tag{4.5}$$

in order to determine the closest segment for a certain grid point $\vec{p}$. Here $\vec{t} = (t_1, \ldots, t_D)$ denotes the unit vector pointing from $\vec{q}$ to the centroid $\vec{c}$ of the segment

$$\vec{t} := \frac{\vec{c} - \vec{q}}{\|\vec{c} - \vec{q}\|} \quad \text{with } \vec{c} := \frac{1}{D} \sum_{i=1}^{D} \vec{x}_i. \tag{4.6}$$

ε_2 is again a small positive constant ($0 < \varepsilon_2 \ll 1$). However, the distance which is finally assigned to $\Phi(\vec{p})$ is still calculated using (4.4). $\varepsilon_2 = 10^{-6}$ is used for all LS initializations in this work.

The initialization procedure is now as follows: For each segment and for all intersecting grid lines all grid points fulfilling (4.3) are determined. The indices $\vec{p}$ of these grid points are appended to a list along with their corresponding distances τ and τ', defined in (4.4) and (4.5), respectively. Finally the list is lexicographically sorted with respect to the grid point indices. If there are more entries with the same $\vec{p}$, (which is not very often the case,) τ of that entry with the smallest corresponding τ' is used for the initial LS value $\Phi(\vec{p})$.

The entire initialization algorithm has a complexity of $\mathcal{O}(N_\mathrm{D} \log N_\mathrm{D} + N_\mathrm{S})$, where N_D is the final number of defined grid grid points in the H-RLE data structure. N_S is the total number of segments of the boundary mesh. The logarithmic term is due to the sorting algorithm which cannot be avoided, since the setup of the H-RLE data structure requires a sorted list as well. This initialization algorithm can produce inefficient sets of active grid points as exemplified in Figure 4.2b, which can be avoided by appending a pruning procedure, as mentioned earlier (see Section 3.4.2).

4.1.2 H-RLE Data Structure Setup

The H-RLE data structure can be set up by inserting all defined grid points in lexicographical order along with their LS values. In this way only the ends of the dynamic arrays must be modified, which can be performed with constant complexity using, for instance, vectors from the standard template library (STL) [124].

The LS framework provides two functions for setting up the H-RLE data structure. The first function allows the insertion of a single defined grid point, while the other one adds a new run of undefined grid points (undefined run) together with a specific run code, indicating the sign of their LS values. Both functions take two parameters. The first parameter is the index vector of the first grid point of the segment. (See Section 3.5.4 for the definition of a segment regarding the H-RLE data structure.) The second parameter takes the LS value for the defined grid point, or the run code for the new run, respectively. By appending successive new defined grid points and new undefined runs in lexicographical order, the H-RLE data structure can be set up in linear time.

For example, the following is done to set up the H-RLE data structure as given in Figure 3.6, for which the corresponding segmentation is shown in Figure 3.7: For each segment, one of the two setup functions must be called. The first segment is the positive undefined run starting at $(0, 0)$. Then, the same function must be called with start indices $(0, 2)$. Next, the second function for inserting defined grid points must be called multiple times with indices $(2, 2), (3, 2), \ldots, (9, 2)$ along with the corresponding LS values. Afterwards, another positive undefined run is inserted which starts at

$(10, 2)$, and so on. Runs are automatically finished, if another new run with a different run code is started or a defined grid point is inserted.

The previously described closest point transform algorithm does not deliver information about the start indices of undefined runs. The output is only a sorted list of all grid points which have at least one neighbor with opposite LS sign. However, from this list and the grid boundaries all starting indices can be derived.

4.2 Sequential Data Access

For an easy sequential access to LS values of grid points stored within a H-RLE data structure iterators have been developed. These iterators can be moved over the data structure in linear time with respect to the number of defined grid points. They are the basis for the realization of algorithms such as the sparse field LS method with optimal linear scaling.

4.2.1 Basic Iterator

The basic iterator traverses the H-RLE data structure in sequential order and stops at every segment, thus at every defined grid point and at every undefined run. The iterator can be moved either forward or backward. Each step requires constant time on average. Since the number of segments is proportional to the number of defined grid points (see Section 3.5.4), a traversal of the entire data structure is possible in linear time.

The iterator allows access to all stored information of the current segment:

- The iterator provides a function that returns the minimum and maximum index vectors of the current segment. If the segment contains only a single grid point, as is the case for defined grid points, both index vectors are identical.
- The LS sign can be queried for the current segment. This is also possible for segments which are undefined runs. In this case the sign can be retrieved from the run code.
- If the current segment is a defined grid point, its LS value can be accessed using the iterator. For undefined runs $+\infty$ or $-\infty$ is returned instead. Floating-point data types fulfilling the IEEE standard for binary floating-point arithmetic [49] provide numerical representations for $+\infty$ and $-\infty$. Alternatively, the maximum and minimum representable value can be used instead.

Figure 4.3b shows a two-dimensional example of a H-RLE data structure with sequentially numbered segments. The numbering corresponds to the traversal sequence of the iterator. The second position refers to the positive undefined run which contains the

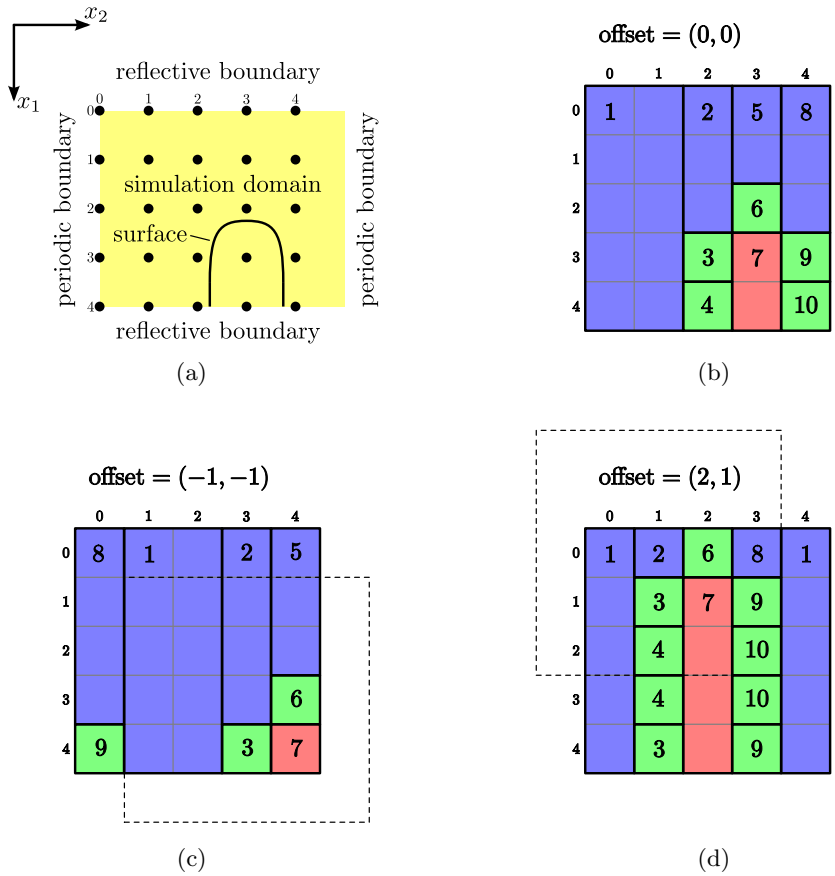


Figure 4.3: (a) A surface embedded in a domain with extensions 4×5. However, due to the different boundary conditions in x_1-direction (reflective) and x_2-direction (periodic) 5×5 grid points are used for the discretization of the level set function. (b) Basic iterators traverse the segments of the H-RLE data structure according to their numbering. (c) The segmentation as seen by an iterator with offset $(-1, -1)$. (d) The same for an iterator with offset $(2, 1)$.

grid points $(0, 2)$, $(1, 2)$, and $(2, 2)$. Hence, in this case the minimum and maximum index vectors are $(0, 2)$ and $(2, 2)$, respectively.

Aside from stepping forward and backward the basic iterator can also be moved to any segment, specified by an index vector belonging to that segment. However, this corresponds to a random access operation with a logarithmic complexity in the worst case.

4.2.2 Offset Iterator

If access to neighbor grid points is needed, additional iterators are used which are moved simultaneously with a central basic iterator over the H-RLE data structure

[44, 84]. Each additional iterator has a predefined offset relative to the central iterator. For example, to enable the calculation of first order differences in two dimensions four additional iterators are necessary with offset index vectors $(\pm 1, 0)$ and $(0, \pm 1)$.

As described earlier the H-RLE data structure is able to store the LS values of a finite set of points of a grid with infinite extensions. However, sometimes it is desireable to limit the simulation domain by using periodic or reflective boundary conditions. In topography simulations, such boundaries are usually used for lateral extensions. (Infinite boundaries are also implemented as reflective boundaries, for which the corresponding minimum and maximum grid indices are set to the limits of the used integral data type.)

Boundary conditions must be incorporated while traversing an offset iterator over the H-RLE data structure. The movement is not as continuous as for the basic iterator, because, due to the boundary conditions, leaps within the data structure may be necessary to reach the next position. At reflective boundaries the moving direction of an offset iterator may also be reversed.

To enhance the usability of iterators an offset iterator has been implemented which provides the same functionality as the basic iterator and which automatically incorporates the applied boundary conditions. It is initialized with a certain offset index vector. The behavior of an offset iterator amounts to a basic iterator moving over the H-RLE data structure, obtained by shifting all segments of the H-RLE data structure according to the (reversed) offset with consideration of the boundary conditions.

Figure 4.3 exemplifies the behavior of offset iterators. Figure 4.3a shows a small surface which is resolved on a simulation domain of size 4×5, with reflective and periodic boundary conditions along the x_1- and x_2-direction, respectively. Storing the LS values of defined grid points using the H-RLE data structure leads to the segmentation as given in Figure 4.3b. Figure 4.3c and Figure 4.3d show the segmentation of the H-RLE data structure as seen for iterators with offsets $(-1, -1)$ and $(2, 1)$, respectively.

The $(-1, -1)$-iterator gives access to the $(-1, -1)$-neighbors of all grid points with indices in the range given by the start and end index vectors returned by the offset iterator. For example, the fourth position of the $(-1, -1)$-iterator refers to the positive undefined run with segment number 2. The minimum and maximum index vector are $(0, 3)$ and $(3, 3)$, respectively, which means that the referenced segment contains the $(-1, -1)$-neighbors for the grid points $(0, 3)$, $(1, 3)$, $(2, 3)$, and $(3, 3)$. A look at Figure 4.3b confirms that for these grid points the $(-1, -1)$-neighbors belong to the positive undefined run with number 2. During a complete traversal of the $(-1, -1)$- and the $(2, 1)$-iterator the segments of the H-RLE data structure are passed in the order $(8, 9, 1, 2, 3, 5, 6, 7)$ and $(1, 2, 3, 4, 4, 3, 6, 7, 8, 9, 10, 10, 9, 1)$, respectively. For reflective boundaries it may not be necessary for all segments to be visited by an offset iterator. For example, the $(-1, -1)$-iterator never stops at segment 4.

A traversal over the entire H-RLE data structure can be performed with linear complexity for all offset iterators, independent of the offset index vector. In most cases,

the next segment can be reached in constant time by stepping forward, similar to the basic iterator. Alternatively, in the case of reflective boundary conditions backward steps can be necessary, as is the case for the movement of the $(2, 1)$-offset iterator from point $(3, 1)$ to $(4, 1)$ (see Figure 4.3d). If the next iterator position is neither the next nor the previous segment, a random access operation is necessary. This is the case, when the next iterator position is on another x_1-grid line (parallel to the x_1-direction) and the first position of the offset iterator must be found there. Due to the offset the first segment is not necessarily the first position.

A random access operation requires a binary search. In order to estimate the additional costs for the binary searches during a complete traversal over the H-RLE data structure all x_1-grid lines, to which at least one defined grid point belongs, are considered. Let Z be the number of these grid lines. The distribution of defined grid points over these Z grid lines is given by the positive numbers $u_1, u_2, \ldots, u_Z$ with $\sum_{z=1}^{Z} u_z = N_\mathrm{D}$. N_D is the total number of defined grid points. Iterators will only traverse these grid lines. In order to find the start position on the z-th grid line a binary search is necessary which has a worst case logarithmic complexity $\mathcal{O}(\log u_z)$. Hence, the total costs during a complete traversal over the H-RLE data structure are $\mathcal{O}(\sum_{z=1}^{Z} \log u_z)$ which can be further estimated as

$$\mathcal{O}(\textstyle\sum_{z=1}^{Z} \log u_z) \leq \mathcal{O}(\sum_{z=1}^{Z} u_z) = \mathcal{O}(N_\mathrm{D}). \tag{4.7}$$

Therefore, the total additional costs due to the binary searches for the worst case scenario are of order $\mathcal{O}(N_\mathrm{D})$, which proves the linear complexity for a complete traversal of offset iterators.

Several offset iterators can be grouped and moved simultaneously over the data structure to enable neighbor access to grid points. As an example, an iterator for first order finite differences in two dimensions can be built from 5 iterators with offsets $(0, 0)$, $(\pm 1, 0)$, and $(0, \pm 1)$. The common set of grid points of all iterators is given by the maximum of all their minimum index vectors and the minimum of all their maximum index vectors. The current iterator positions only apply for grid points within this range. To go to the next grid points the common range must be moved forward. This is performed by advancing only those iterators for which their maximum index vector is equal to the upper bound of the common range. In this way an iteration over the H-RLE data structure can be carried out in linear time. However, by nature, the costs are directly proportional to the number of iterators in the group. Due to the inherent incorporation of boundary conditions arbitrary stencils of iterators can be easily formed. This allows a simple implementation of algorithms for the H-RLE data structure.

4.3 Sparse Field Implementation

The H-RLE data structure does not support efficient insertion or deletion of defined grid points except at the end. However, the sparse field LS method changes the set of defined grid points at every time step. The only efficient way to get a new H-RLE data structure with a different set of defined grid points is to set it up from the beginning. This can be carried out on-the-fly by simultaneously iterating over the existing H-RLE data structure and inserting the desired defined grid points into the new data structure.

In the following sections the realization of the sparse field LS method for the H-RLE data structure is described in further detail. Every time step requires the application of a couple of operations which all process the H-RLE data structure sequentially using the previously described iterators. As a consequence, all these operations exhibit a linear performance.

The sparse field implementation extends the original H-RLE data structure by one additional array which has the same size as the array holding the LS values. For each defined grid point an integral value is stored, which gives information on whether a defined grid point is active or not. In the first case, the corresponding value gives the number of the active grid point. The numbering corresponds to the lexicographical order of active grid points. By contrast, non-active defined grid points are indicated by setting the corresponding array entries to the maximum representable value.

4.3.1 Time Integration

For the time integration step it is assumed that the defined grid points of the H-RLE data structure include at least all active grid points as well as all neighbors which are needed for the applied time integration scheme. Furthermore, at each active grid point the surface velocity field must be known. To update the LS values of all active grid points in time, first the time increment Δt needs to be determined according to (3.33). This requires an iteration over the data structure using a finite difference stencil of iterators. At each active grid point the approximations to the Hamiltonian are calculated and stored in a separate array, because they are needed again, after the determination of the time increment, for updating the LS values.

After updating the LS values of all active grid points, the LS values of all other defined grid points are no longer needed. Hence, it is favorable to reduce the set of defined grid points. This can be performed during the same initial iteration. Every active grid point is inserted together with its LS value into a new H-RLE data structure. The signs of all non-active grid points are maintained and stored within the new H-RLE data structure as members of undefined runs. After the iteration, the memory used for storing the old H-RLE data structure can be freed.

Once the time increment is known and the approximations to the Hamiltonian are calculated for all active grid points, it is possible to update the LS values of all active grid points using (3.5). This operation could be realized by processing the data structure using a basic iterator. However, since the new H-RLE data structure only contains active grid points, the array holding the LS values has the same size as the array for the approximations to the Hamiltonian. Therefore, the update of the LS values can simply be implemented by a multiplication of the Hamiltonian array with the time increment Δt followed by a subtraction from the LS values array of the H-RLE data structure.

4.3.2 Pruning and Consistency Check

So far the H-RLE data structure contains the updated LS values $\Phi^{(t+\Delta t)}(\vec{p})$ for all grid points from the active layer $\mathcal{L}_0^{(t)}$. As described in Section 3.4.2 a pruning procedure is necessary to avoid dense sets of active grid points. Again, a stencil of iterators, similar to that required for the computation of first order finite differences, is moved over the data structure. At the same time a new H-RLE data structure is set up in which all changes are stored. The new H-RLE data structure is constructed by copying the old one, while skipping defined grid points which do not have an opposite signed neighbor. These grid points are added to undefined runs instead.

Since the neighbors of active grid points (belonging to $\mathcal{L}_{\pm 1}^{(t)}$) have not been updated yet, this early pruning procedure works only if the signs of non-active grid points ($\notin \mathcal{L}_0^{(t)}$) are not altered during the entire time integration step. According to Section 3.4.2 the signs of non-active grid points are maintained, if the CFL number C_{CFL} fulfills

$$C_{\mathrm{CFL}} \leq \frac{1}{2}. \tag{4.8}$$

As a consequence, the presented implementation of the sparse field method only allows CFL numbers satisfying this condition.

Before a defined grid point is inserted into the new H-RLE data structure, it is first checked, if its LS value is greater than $\frac{1}{2}$ or smaller than $-\frac{1}{2}$, while that of any neighboring defined grid point is less than $-\frac{1}{2}$ or larger than $\frac{1}{2}$, respectively. If this is the case, the prerequisite of the sparse field LS method (3.27) would be violated. To guarantee the robustness of the algorithm the LS value must be reduced to $\pm\frac{1}{2}$ (while keeping its sign) before insertion into the new H-RLE data structure. After completing the iteration over the old H-RLE data structure and finishing the setup of the new one, the old H-RLE data structure can be deleted.

4.3.3 Dilation

The set of defined grid points in the H-RLE data structure is as a result of the pruning procedure a subset of $\mathcal{L}_0^{(t)}$. The LS values are all in the range $[-1, 1]$ due to restriction

(4.8). For the next time step the LS values of all new active grid points $\vec{p} \in \mathcal{L}_0^{(t+\Delta t)}$ must be known. Furthermore, the LS values of neighboring layers, as needed for the time integration scheme, must be updated. This requires a dilation procedure which extends the set of defined grid points and calculates their corresponding LS values using the update scheme (3.29).

To extend the set of defined grid points on the positive and the negative side by a single layer, a stencil of iterators, which allows access to neighbor points, is moved over the H-RLE data structure. If any iterator from the stencil encounters a defined grid point, a defined grid point is inserted into the new H-RLE data structure. If the position of the central iterator represents a defined grid point, its LS value is simply copied. Otherwise, the LS value must be determined using the update scheme (3.29) and the LS values of neighbor points, which can be accessed by the non-central iterators. Here, positive and negative undefined runs are regarded to have LS values $+\infty$ or $-\infty$, respectively. Since at least one iterator is at the position of a defined grid point the LS value of the newly inserted grid point is always defined. After completing the iteration, the old H-RLE data structure can be deleted. The new H-RLE data structure now contains all previously defined grid points along with the newly added defined grid points.

After the first application of the dilation procedure, the LS values of all grid points in layers $\mathcal{L}_{\pm 1}^{(t)}$ are updated. Since the CFL condition (3.31) implies that

$$\mathcal{L}_0^{(t+\Delta t)} \subseteq \left(\mathcal{L}_{-1}^{(t)} \cup \mathcal{L}_0^{(t)} \cup \mathcal{L}_1^{(t)} \right), \tag{4.9}$$

the updated LS values are available for all new active grid points $\vec{p} \in \mathcal{L}_0^{(t+\Delta t)}$. Similarly, a second run of the dilation procedure which updates all points in layers $\mathcal{L}_{\pm 2}^{(t)}$ ensures that at least for all grid points $\vec{p} \in \mathcal{L}_{\pm 1}^{(t+\Delta t)}$ the updated LS values are available, and so on. The number of repetitions of the dilation procedure depends on how many additional layers of defined grid points are needed for the time integration scheme. As an example, for the second order approximation, which requires two additional layers, three iterations are necessary in total.

However, this procedure also adds defined grid points to the H-RLE data structure, which are not really needed for the finite difference scheme. If N_L additional layers are necessary, the last iteration which updates all points of layers $\mathcal{L}_{\pm(N_\mathrm{L}+1)}^{(t)}$ also adds some new points which are actually members of $\mathcal{L}_{\pm(N_\mathrm{L}+1)}^{(t+\Delta t)}$. The insertion of these grid points can easily be avoided according to (3.26), if only grid points for which the new LS value fulfills $|\Phi^{(t+\Delta t)}(\vec{p})| \leq N_\mathrm{L} + \frac{1}{2}$ are added. The procedure can be further optimized by inserting new grid points as late as possible to speed up successive iterations over the H-RLE data structure. It is favorable to add only defined grid points with an absolute LS value not larger than $k - \frac{1}{2}$ during the k-th iteration.

The dilation procedure scales with $\mathcal{O}(N_\mathrm{D} \cdot {N_\mathrm{L}}^2)$, where N_D is the number of defined grid points before dilation and N_L is the number of added layers. Hence, this algorithm

is only efficient for small N_L, which is the case, if first ($N_\mathrm{L} = 1$) or second order finite difference schemes ($N_\mathrm{L} = 2$) are used. For larger N_L, an algorithm similar to the fast marching method [109] could probably be the better choice. However, this technique would require random access to the H-RLE data structure resulting in a non-linear performance with respect to N_D.

The pruning and consistency check, as described in the previous section, can be included during the first dilation cycle. This avoids an iteration over the H-RLE data structure and accelerates the sparse field method.

4.4 Boolean Operations

If geometries are represented as level sets, Boolean operations can be expressed as simple operations on the corresponding LS functions [89, 110]. If one considers the sets $\mathcal{M}_A$, $\mathcal{M}_B$, and $\mathcal{M}_C$ and the corresponding boundaries $\mathcal{S}_A = \partial\mathcal{M}_A$, $\mathcal{S}_B = \partial\mathcal{M}_B$, and $\mathcal{S}_C = \partial\mathcal{M}_C$, which are represented by the LS functions Φ_A, Φ_B, and Φ_C, the LS counterparts of Boolean operations are listed in the following.

$$\begin{aligned}
\text{Union:} \quad & \mathcal{M}_A = \mathcal{M}_B \cup \mathcal{M}_C \quad \Leftrightarrow \quad \Phi_A = \min(\Phi_B, \Phi_C) && (4.10)\\
\text{Intersection:} \quad & \mathcal{M}_A = \mathcal{M}_B \cap \mathcal{M}_C \quad \Leftrightarrow \quad \Phi_A = \max(\Phi_B, \Phi_C) && (4.11)\\
\text{Complement:} \quad & \mathcal{M}_A = \mathbb{R}^D \setminus \mathcal{M}_B \quad \Leftrightarrow \quad \Phi_A = -\Phi_B && (4.12)\\
\text{Relative complement:} \quad & \mathcal{M}_A = \mathcal{M}_B \setminus \mathcal{M}_C \quad \Leftrightarrow \quad \Phi_A = \max(\Phi_B, -\Phi_C) && (4.13)
\end{aligned}$$

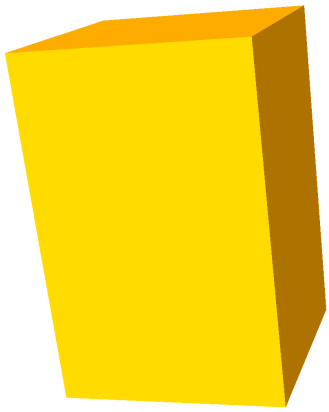
Here the convention is used that the LS function Φ_X is negative for all $\vec{x} \in \mathcal{M}_X$.

Boolean operations are very useful for more general topography simulations, if consecutive process steps, like etching and/or deposition processes, should be simulated or several materials are involved.

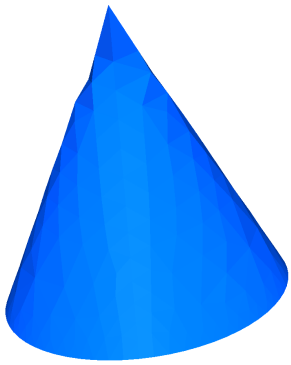
4.4.1 Implementation

If the H-RLE data structure is used, Boolean operations can be implemented in a very efficient manner. The minimum or maximum of two LS functions is obtained by moving two basic iterators over the corresponding H-RLE data structures. The common range of both iterators is defined as the maximum of both minimum index vectors and by the minimum of both maximum index vectors. To advance the pair of iterators to the next position, all iterators are moved forward, whose maximum index vector is equal to the maximum index vector of the common range. This is in analogy to the realization of neighbor access using a stencil of iterators, as described in Section 4.2.2.

At every step, the minimum or the maximum LS value of both iterators is calculated. Recalling that the LS value of undefined runs is $+\infty$ or $-\infty$ (see Section 4.2.1), the

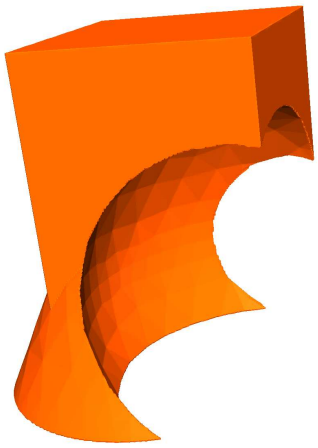


(a) Cuboid $\mathcal{M}_A$.



(b) Cone $\mathcal{M}_B$.

(c) Sphere $\mathcal{M}_C$.



(d) Resulting geometry $(\mathcal{M}_A \cup \mathcal{M}_B) \setminus \mathcal{M}_C$.

Figure 4.4: Boolean operations can be calculated using level sets. The union of a cuboid (a) and a cone (b) subtracted by a sphere (c) is the structure shown in (d).

result can also take these values. In this case an undefined run is inserted into the new H-RLE data structure using the minimum index vector of the common range. Otherwise, if the result is not $+\infty$ or $-\infty$, at least one of the two iterators must be at the position of a defined grid point. Hence, the common range is exactly this single grid point, for which the common minimum and maximum index vector are identical. This index vector is used to add a defined grid point to the H-RLE data structure.

The resulting new H-RLE data structure does not necessarily fulfill (3.30). Hence, grid points belonging to $\mathcal{L}_0$ do not necessarily have an opposite signed neighbor leading to inefficient sets of active grid points. To avoid dense sets of such nonrelevant grid points they must be removed using a pruning procedure as described in Section 4.3.2.

Overall, the calculation of the minimum or the maximum of two LS functions Φ_A and Φ_B can be carried out in linear time $\mathcal{O}(N_{\mathrm{D}}^{(A)} + N_{\mathrm{D}}^{(B)})$, if $N_{\mathrm{D}}^{(A)}$ and $N_{\mathrm{D}}^{(B)}$ denote the corresponding numbers of defined grid points. The determination of the complement requires only the inversion of the sign. This can be performed without rebuilding the H-RLE data structure, by simply changing the sign of all values in the LS array and interchanging the run codes of all positive and negative undefined runs. The application of Boolean operations on H-RLE data structures is demonstrated in Figure 4.4. It shows the relative complement of a sphere in the union of a cuboid and a cone.

4.4.2 Chemical-Mechanical Planarization

If a geometry is represented as LS, Boolean operations can easily be calculated. They can be applied to simulate various semiconductor processes in a simplified manner. For example, the simplest model for chemical-mechanical planarization (CMP) assumes that everything is cut away above a certain height. Hence, the new structure $\mathcal{M}'$ is obtained by

$$\mathcal{M}' = \mathcal{M} \cap \{\vec{x} : \vec{x} \cdot \vec{n} \leq c\}, \tag{4.14}$$

where $\vec{n}$ is the normal vector and c the distance to the origin of the cutting plane $\mathcal{P}_{\mathrm{cmp}} = \{\vec{x} : \vec{x} \cdot \vec{n} = c\}$. By setting up a LS function Φ_{cmp}, whose zero LS represents $\mathcal{P}_{\mathrm{cmp}}$ (4.14) can be written as

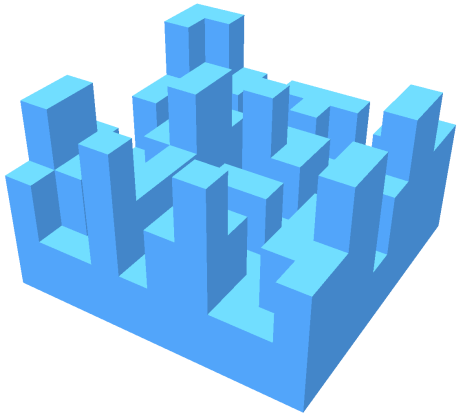
$$\Phi' = \max(\Phi, \Phi_{\mathrm{cmp}}). \tag{4.15}$$

Here Φ and Φ' are the LS functions describing $\mathcal{M}$ and $\mathcal{M}'$, respectively.

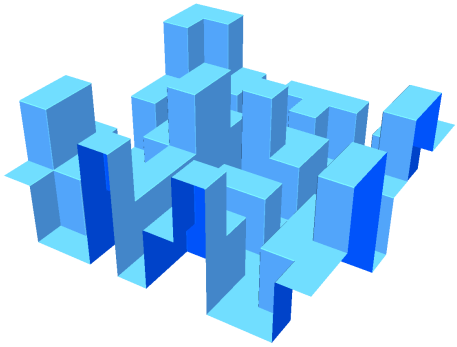
Figure 4.5 shows an example, for which CMP is used to flatten the geometry after an isotropic deposition process. Boolean operations are applied to both, the LS function representing the initial structure and the LS function representing the final structure after the deposition process.

4.4.3 Pattern Transfer

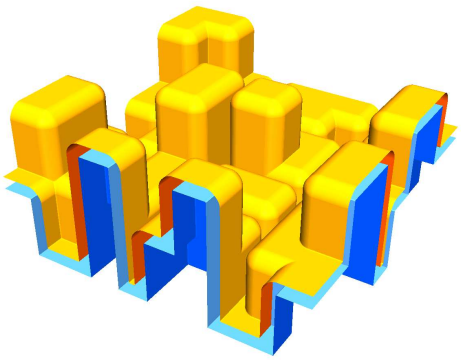
Boolean operations can also be used to transfer a pattern defined by a given mask onto a structure. It is assumed that the initial geometry is a two-layer structure like that shown in (Figure 4.6a). The geometry can be alternatively described by two LS functions (Figure 4.6b). The first LS function Φ_1 represents the interface between the two layers and the second Φ_2 describes the surface. To transfer the mask pattern to



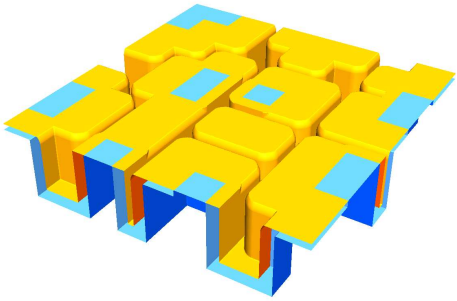
(a) The initial geometry with a bounding box $500 \times 500 \times 300$ measured in grid spacings.



(b) The corresponding LS representation.



(c) Isotropic deposition.



(d) Chemical-mechanical planarization.

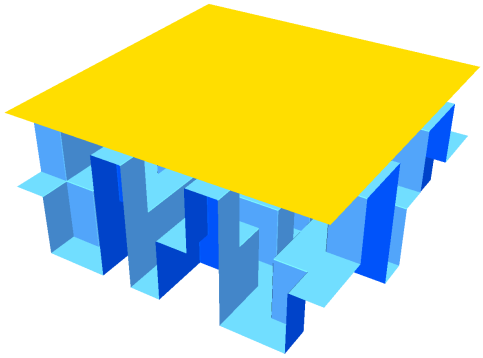
Figure 4.5: After an isotropic deposition process the structure is exposed to CMP. This process is realized using Boolean operations applied on the LS representations of the initial surface (blue) and the surface after the deposition (yellow).

the structure the top layer needs to be selectively removed. This can be expressed by the following Boolean operation

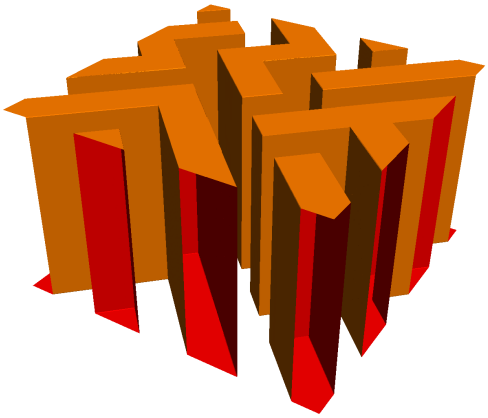
$$\Phi_2' = \min(\Phi_1, \max(\Phi_2, \Phi_{\text{mask}})). \tag{4.16}$$

Here the LS function Φ_{mask} represents the mask, if it is extruded to the third dimension (Figure 4.6c). The final minimum and maximum coordinates of the extruded mask must be at least smaller and larger than the minimum and maximum coordinates of the LSs representing the initial structure, respectively. The final structure is shown in Figure 4.6d.

(a) The mask (brown) defines the pattern which is finally transfered onto the initial structure (blue) by partially removing the resist (yellow).



(b) The LS representation of the two-layer structure.



(c) The mask is extruded to three dimensions and also represented as LS.

(d) The final structure with the transfered pattern.

Figure 4.6: Boolean operations using level sets can be applied to describe pattern transfer.

4.5 Smoothing

The LS method also provides a simple way to smooth a given geometry. Setting the surface velocity in the LS equation equal to the mean curvature leads to a smoothed surface [110]. The definition of the mean curvature and its approximation were given in Section 3.3.2. The introduction of a lower limit $\kappa_{\min} < 0$ and an upper limit $\kappa_{\max} > 0$ for regions with negative and positive curvature, respectively, controls the amount of smoothing. The smoothing algorithm is realized by setting the surface velocity field

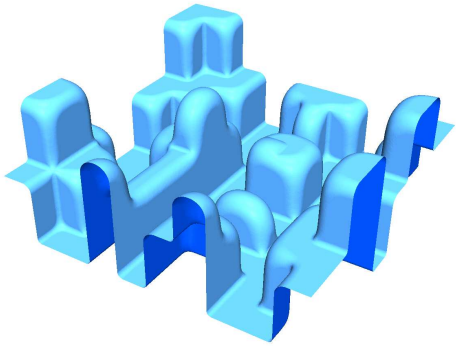


Figure 4.7: The final surface after a smoothing operation is applied to the geometry given in Figure 4.5a. The curvature is limited by $\kappa_{\min} = -0.1$ and $\kappa_{\max} = 0.05$.

as follows

$$V(\vec{p}) = \begin{cases} 0 & \text{if } \kappa_{\min} \leq \kappa(\vec{p}) \leq \kappa_{\max}, \\ -\kappa(\vec{p}) & \text{else,} \end{cases} \tag{4.17}$$

and solving the LS equation over time until all surface velocities are equal to 0. Figure 4.7 demonstrates smoothing for the test structure given in Figure 4.5a with $\kappa_{\min} = -0.1$ and $\kappa_{\max} = 0.05$.

4.6 Multiple Material Regions

General topography simulations require the treatment of different material regions (see Section 2.1). Etch processes especially require the distinction of different material regions, such as a mask and a substrate, to which different etch rates have to be applied. Hence, the geometric information of material regions is necessary during time evolution. Usually the initial geometry is given as a triangulated mesh, where each element is assigned to a certain material. In the case of etching, this irregular grid must be accessed many times to query the material region of surface points in order to calculate the correct surface velocities. Therefore, if search trees are used, for which queries are of logarithmic complexity, a linearithmic algorithm for time evolution can be expected. If consecutive deposition and etching processes must be simulated, a costly and challenging modification of the irregular mesh is necessary after each processing step.

Instead of using an irregular mesh for the material regions and a regular grid for the LS method simultaneously, multiple LS functions can be used instead. The initial geometric information is simply mapped from the irregular mesh to the regular grid by means of additional LS functions. In the following sections a multi-LS technique using the sparse field method and the H-RLE data structure is described. With the ability to

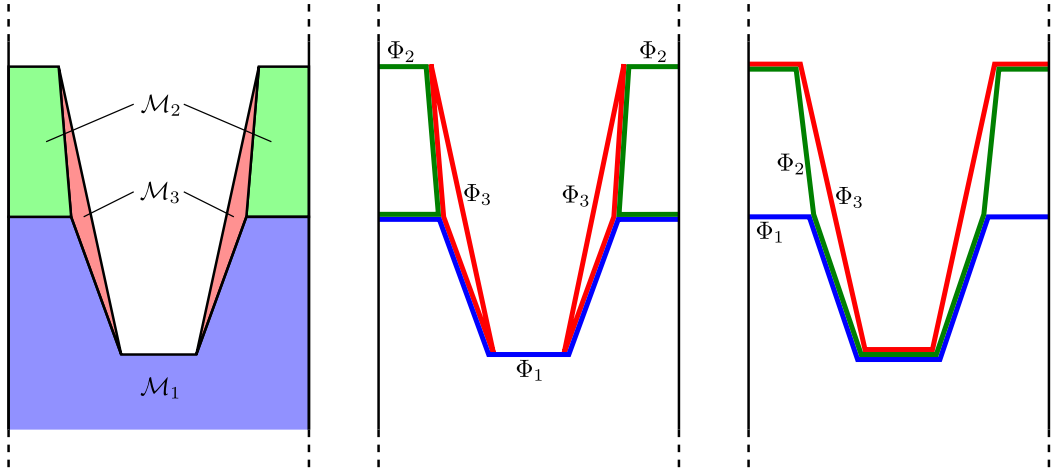


Figure 4.8: (a) A geometry consisting of three different materials, where $\mathcal{M}_1$ represents the substrate, $\mathcal{M}_2$ the mask, and $\mathcal{M}_3$ a passivation layer. (b) Description of the structure using three enclosing LSs. (c) Alternative representation where one LS describes the common surface and two others the interfaces between the different material regions.

resolve the material-dependent surface velocities with sub-time-step accuracy, a more accurate final profile is obtained, especially in the presence of thin layers or large etch rate ratios.

4.6.1 Level Set Representation

For the following considerations the entire structure $\mathcal{M}$ is assumed to be composed of M disjoint material regions $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_M$ satisfying (2.1). There are several possibilities to represent the different material regions by LS functions. One way is to describe each material region $\mathcal{M}_k$ by an enclosing LS function Φ_k [46]

$$\Phi_k(\vec{x}) \leq 0 \quad \Leftrightarrow \quad \vec{x} \in \mathcal{M}_k \tag{4.18}$$

as shown in Figure 4.8b. However, using this representation, very thin layers with thicknesses smaller than one grid spacing cannot be properly resolved. However, some etching processes require the treatment of very thin passivation layers. If the passivation layer becomes thinner than the grid spacing, it can vanish abruptly, and the etching of the underlying material would start too early. This can lead to significant errors, especially for large etch rate ratios.

Instead, different material regions can be described by M LS functions satisfying

$$\Phi_k(\vec{x}) \leq 0 \quad \Leftrightarrow \quad \vec{x} \in \bigcup_{i=1}^{k} \mathcal{M}_i. \tag{4.19}$$

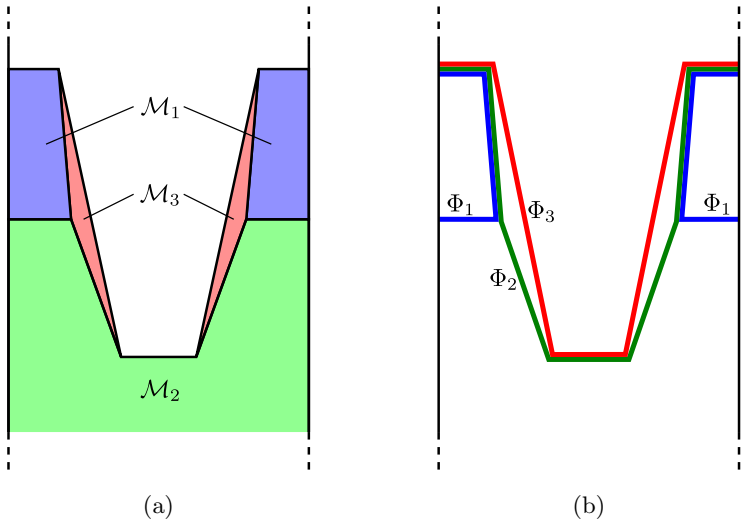


Figure 4.9: Renumbering of material regions (a) leads to a different LS representation (b).

Here Φ_M describes the surface of the entire structure $\mathcal{M}$ and the other LS functions correspond to interfaces as depicted in Figure 4.8c. This LS representation is often more convenient for topography simulation, because thin layers, such as the passivation layer $\mathcal{M}_3$ in Figure 4.8a, are described more accurately.

The LS configuration according to (4.19) depends on the numbering of the material regions. For comparison, a different LS representation, obtained through relabeling of the material regions is shown in Figure 4.9. The most suitable LS representation depends on the problem. If the mask is completely removed during the etch process, the configuration given in Figure 4.8c might be better. If underetching of the mask is expected, the LS representation shown in Figure 4.9b is favorable.

If the geometry is described using (4.19), the number of the material region which is on the surface can easily be determined using

$$\mathrm{mat}(\vec{x}) := \min\{1 \leq k \leq M : \Phi_k(\vec{x}) = 0\}. \tag{4.20}$$

mat is the surface material function which is defined for all surface points $\vec{x} \in \mathcal{S}$.

For the following considerations all LS functions are assumed to fulfill

$$\Phi_1(\vec{x}) \geq \Phi_2(\vec{x}) \geq \ldots \geq \Phi_M(\vec{x}). \tag{4.21}$$

This inequality is inherently satisfied, if the geometry is described by (4.19) and the LS functions are initialized as distance functions.

Some processes deposit new material layers on top of the structure. The described LS representation allows for easy additions of new material regions. It is sufficient to introduce a new LS function Φ_{M+1}, which satisfies $\Phi_{M+1}(\vec{x}) \leq \Phi_M(\vec{x})$. In practice,

to model the deposition of a new material, the new LS function is first initialized according to $\Phi_{M+1}(\vec{x}) = \Phi_M(\vec{x})$. Then the LS equation is solved for this topmost LS function Φ_{M+1}, which represents the new surface. Stripping the top most material layer $\mathcal{M}_M$ is also very simple by deleting the top most LS function $\Phi_M(\vec{x})$.

4.6.2 Time Evolution

The different material regions must be taken into account during time evolution, especially for the simulation of etching. Common approaches use a single velocity field which is set up in dependence of the material types on the surface [110]. This field is assumed to be constant for the duration of the time integration step. However, if the etch front reaches another material within that time step, for which the value of the etch rate differs significantly, as is the case for mask or etch stop layers, the surface is advanced with the wrong velocity. For the LS representation introduced in (4.19) a more accurate technique is presented in this section, which is able to resolve varying surface velocities with sub-time-step accuracy.

Initially only the topmost LS function Φ_M, which represents the surface, is updated in time. Thereby, the surface velocities of different materials which are on the surface and which can be retrieved from the surface material function (4.20) are incorporated. In the following, it is always assumed that material of type $\mathcal{M}_M$ is deposited, if the surface velocities are positive. If another material should be deposited instead, a new material layer must be added, as described previously. Negative surface velocities stand for material removal, and the material dependence on the surface velocity must be incorporated during the time evolution of Φ_M.

After updating the topmost LS function Φ_M in time, all other LS functions representing the interfaces between material regions, are adapted according to the Boolean operation

$$\Phi_k^{(t+\Delta t)}(\vec{x}) = \max\left(\Phi_k^{(t)}(\vec{x}), \Phi_M^{(t+\Delta t)}(\vec{x})\right). \tag{4.22}$$

This adaption rule maintains relation (4.21). For pure deposition processes, where the surface velocity is positive everywhere on the surface, (4.22) does not change the other LS functions. However, if there are any regions on the surface with negative surface velocities, these Boolean operations must be performed.

There is still the open question of how to change Φ_M under consideration of the different material regions with sub-time-step accuracy. For simplicity M different velocity fields $V_k(\vec{x})$ $(1 \leq k \leq M)$ are introduced, one for each material type. The value of $V_k(\vec{x})$ at a certain surface point $\vec{x} \in \mathcal{S}$ is defined as the surface velocity, which would be obtained, if material $\mathcal{M}_k$ was on the surface at that point, i.e. $\mathrm{mat}(\vec{x}) = k$ holds. However, $V_k(\vec{x})$ is still calculated using the arriving flux distribution $\Gamma(\vec{x}; q, \vec{\omega}, E)$. This distribution can be assumed to be constant for the duration of the time integration step, because small changes of the geometry usually do not show much influence on

particle transport. Even if the material type changes on small parts of the surface during the time step, which leads to different reemission probabilities, the flux distribution remains approximately the same.

Due to the convention that only the topmost material $\mathcal{M}_M$ can be deposited, the surface velocity fields must obey

$$V_k(\vec{x}) \in \mathbb{R}_0^- \qquad \text{for } 1 \leq k < M, \tag{4.23}$$

$$V_M(\vec{x}) \in \mathbb{R}. \tag{4.24}$$

The time evolution of Φ_M is computed using a generalization of the update rule (3.5)

$$\Phi_M^{(t+\Delta t)}(\vec{p}) = \Phi_M^{(t)}(\vec{p}) - \sum_{k=1}^{M} \Delta t_k(\vec{p}) \cdot \hat{H}(\vec{p}, \Phi_M^{(t)}, V_k(\vec{p})), \tag{4.25}$$

which is applied to all active grid points $\vec{p} \in \mathcal{L}_0^{(t,M)}$. Here, in accordance with the sparse field method, $\mathcal{L}_0^{(t,M)}$ denotes the active layer of Φ_M at time t. $\Delta t_k(\vec{p})$ denotes the time for which the surface velocity $V_k(\vec{p})$ is used during the time integration of the LS value at point $\vec{p}$. The sum of all these times must satisfy $\sum_{k=1}^{M} \Delta t_k(\vec{p}) = \Delta t$. To incorporate material-dependent surface velocities with sub-time-step accuracy they are approximated for a given time step Δt using

$$\Delta t_k(\vec{p}) = \begin{cases} \frac{\Phi_k^{(t)}(\vec{p}) - \Phi_{k-1}^{(t)}(\vec{p})}{\hat{H}(\vec{p}, \Phi_M^{(t)}, V_k(\vec{p}))} & \text{if } k > k'(\vec{p}), \\ \Delta t - \sum_{l=k+1}^{M} \Delta t_l(\vec{p}) & \text{if } k = k'(\vec{p}), \\ 0 & \text{if } k < k'(\vec{p}). \end{cases} \tag{4.26}$$

For each active grid point $\vec{p}$ the material number $k'(\vec{p})$ is chosen to be the smallest number ($1 \leq k'(\vec{p}) \leq M$) for which all $\Delta t_k(\vec{p})$ are non-negative, if they are calculated according to (4.26). If $V_M(\vec{p})$ is positive, $k'(\vec{p})$ is always set to M. $k'(\vec{p})$ can be interpreted as the material type which is locally on the surface after the time integration step. Insertion of (4.26) into (4.25) yields

$$\Phi_M^{(t+\Delta t)}(\vec{p}) = \Phi_{k'}^{(t)}(\vec{p}) - \Delta t_{k'}(\vec{p}) \cdot \hat{H}(\vec{p}, \Phi_M^{(t)}, V_{k'}(\vec{p})). \tag{4.27}$$

It should be noted that it is not necessary to evaluate the velocity fields V_k for all k and at all active grid points $\vec{p} \in \mathcal{L}_0^{(t,M)}$. For an active grid point $\vec{p}$ it is sufficient to calculate $V_k(\vec{p})$ for $k'(\vec{p}) \leq k \leq M$. $V_k(\vec{p})$ with $k < M$ is also not relevant, if $\Phi_k^{(t)}(\vec{p}) = \Phi_{k-1}^{(t)}(\vec{p})$. Moreover, if $V_M(\vec{x})$ is never positive, due to the applied model, then the determination of $V_M(\vec{p})$ can also be omitted in the case of $\Phi_M^{(t)}(\vec{p}) = \Phi_{M-1}^{(t)}(\vec{p})$. Hence, for a certain active grid point only the local surface velocities for those materials, which are actually involved during the time step, must be determined.

The time integration step Δt is chosen according to (3.32) in such a way that

$$\max_{\vec{p} \in \mathcal{L}_0^{(t,M)}} \left| \Phi_M^{(t+\Delta t)}(\vec{p}) - \Phi_M^{(t)}(\vec{p}) \right| = C_{\mathrm{CFL}} \tag{4.28}$$

is satisfied. Due to the limitation $C_{\mathrm{CFL}} \leq \frac{1}{2}$ (4.8) it is sufficient for the solution of (4.26) to store only LS values up to an absolute value of 1 within the H-RLE data structures for the LS functions $\Phi_1, \Phi_2, \ldots, \Phi_{M-1}$. LS values of positive and negative undefined grid points are again considered to be equal to $+\infty$ and $-\infty$, respectively (compare Section 4.2.1).

The presented multi-LS method can be realized by adapting the time integration procedure described in Section 4.3.1. While iterating over the surface LS function Φ_M using a stencil of iterators, which allows the calculation of the required finite differences, $M-1$ additional basic iterators are simultaneously moved over the corresponding H-RLE data structures of $\Phi_1, \Phi_2, \ldots, \Phi_{M-1}$. These iterators allow access to the LS values as needed in (4.26).

4.6.3 Isotropic Material Dependent Etching

To demonstrate the multi-LS approach, an isotropic etching process is applied to the test structure given in Figure 4.10. The lateral extensions of this geometry are 1200×200 (in terms of grid spacings). The structure consists of three layers on top of a substrate. The layer thicknesses are, from top down, 50, 50, and 0.5. The top layer is a mask with four circular holes with diameters 20, 40, 60, and 80, respectively.

The way in which the different material regions are labeled defines the LS representation according to (4.19). Since underetching of the mask is expected, the mask is assigned to $\mathcal{M}_3$. The complete numbering can be seen in Figure 4.10. The corresponding LS representation of the initial structure is depicted in Figure 4.11. Four LS functions are used to describe the four different material regions.

The multi-layer structure is exposed to an etch process. The etch rate is assumed to be 0.1 grid spacings per time unit for the mask ($\mathcal{M}_3$) and 0.025 for the very thin layer ($\mathcal{M}_2$). For the other two material regions the etch rate is set to 1. The profile after 120 time units is shown in Figure 4.12. Although the very thin layer has a thickness smaller than a grid spacing, its small etch rate is accurately incorporated during time evolution.

In order to prove the linear scaling of the entire time evolution algorithm, this etch process is applied to the same geometry, only scaled by various factors. The process times are scaled accordingly. Table 4.1 lists the average computation times for a single time step for scale factors 0.5, 1.0, 1.5, 2.0, and 2.5. The calculation time scales well with the surface size which itself scales quadratically with the scale factor. The total number of required time steps is also given. The numbers are obtained using

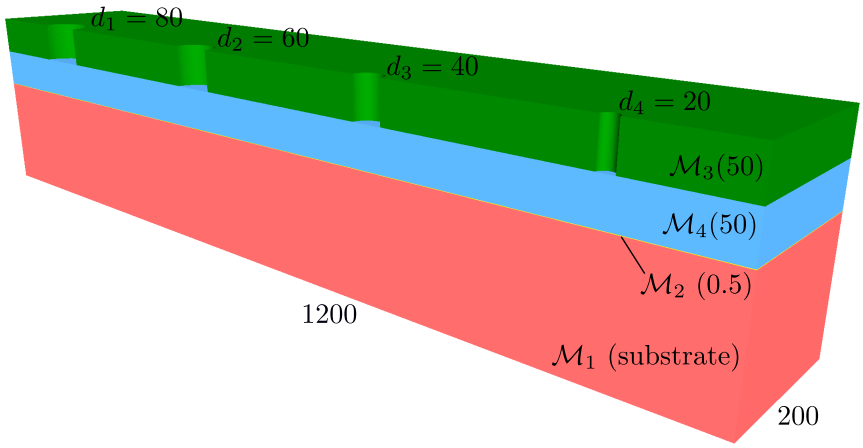


Figure 4.10: A test structure with lateral extensions 1200×200 and three layers on top of the substrate (red). The first layer cannot be clearly seen, since it has a thickness of only 0.5. The second layer (blue) and the mask (green) both have a thickness of 50.

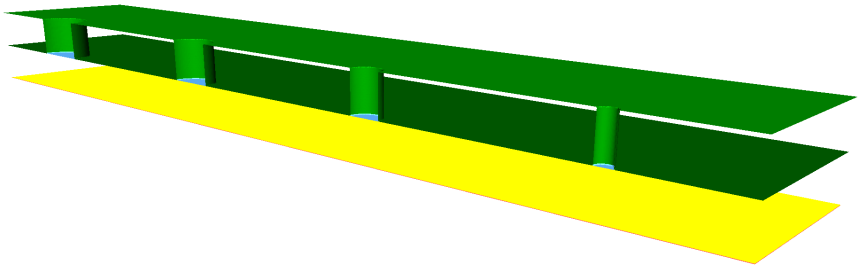


Figure 4.11: The multi-LS representation of the initial geometry. Four LS functions are used to describe the four different material regions.

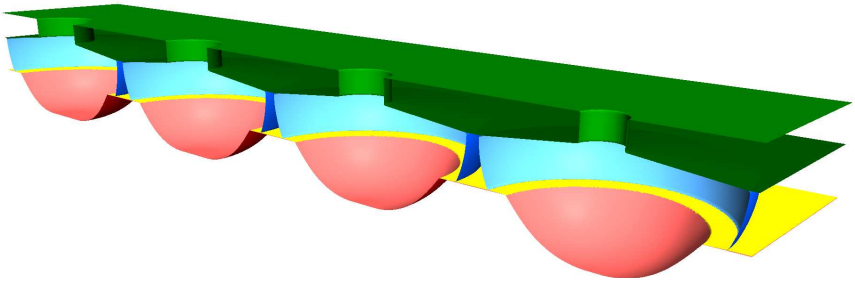


Figure 4.12: The final geometry after isotropic etching with material-dependent etch rates. The etch rate was 0.1 grid spacings per time step for the mask, and 0.025 for the very thin layer. For the other two material regions the etch rate was set to 1.

Scale factor	0.5	1.0	1.5	2.0	2.5
Lateral grid resolution	600×100	1200×200	1800×300	2400×400	3000×500
Time integration (avg.)	0.38 s	1.56 s	3.44 s	6.10 s	9.81 s
Num. time steps	1002	2035	3070	4104	5139

Table 4.1: The average computation time for a time integration step and the number of required time steps for the material-dependent etching process shown in Figure 4.12 using an AMD Opteron 8222 SE processor (3 GHz).

$C_{\mathrm{CFL}} = 0.5$. The simulation was carried out on an AMD Opteron 8222 SE processor with a CPU clock speed of 3 GHz.

4.7 Directional Visibility Check

Some processes are characterized by almost unidirectional and normal incident particle transport from the source plane to the surface. In this case the H-RLE data structure can be useful for the computation of surface velocities for the active grid points. The surface velocity field at active grid points is usually obtained by taking the surface velocity of the closest surface point. The surface velocity of a surface point depends on the visibility from the particle source along the incidence direction. Instead of determining the visibilities for surface points and mapping the information to the active grid points, it is possible to compute the visibility information directly for all active grid points.

In the following section, an active grid point is called visible, if the corresponding closest surface point is visible. Furthermore, it is assumed that the incidence direction is equal to the positive x_1-direction. Then, an active grid point $\vec{p}$ can be regarded as visible, if the LS values of all grid points $\vec{p}' = \vec{p} - k \cdot \vec{e}_1$ with $k > 0$ are larger than or equal to that of grid point $\vec{p}$

$$\vec{p} \text{ is visible} \quad \Leftrightarrow \quad \forall k > 0 : \Phi(\vec{p}) \leq \Phi(\vec{p} - k \cdot \vec{e}_1). \tag{4.29}$$

Thereby, the LS values of positive and negative undefined grid points are again assumed to be $\pm\infty$, respectively. Figure 4.13 illustrates by means of an example, which active grid points are visible. Taking advantage of the lexicographical ordering, it is sufficient to iterate once over the H-RLE data structure to determine the visibilities for all active grid points. Hence, this visibility test is of optimal complexity $\mathcal{O}(N_{\mathrm{D}})$, where N_{D} denotes again the number of defined grid points, which is proportional to the surface size measured in grid spacings.

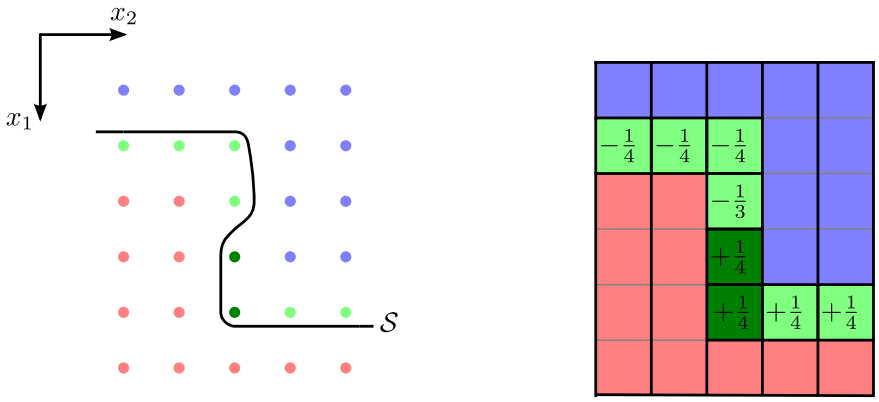


Figure 4.13: A surface $\mathcal{S}$ and its H-RLE representation are shown. In this example, the LS values are only defined for the active grid points. Positive (blue) and negative (red) runs of undefined grid points are assumed to have LS values $+\infty$ and $-\infty$, respectively. The dark green grid points do not fulfill the visibility criterion.

4.7.1 Directional Etching

For a visible grid point the local flux on the surface can be computed as

$$F(\vec{p}) = F^{\text{src}} \cdot \max(0, -n_1(\vec{p})), \tag{4.30}$$

where F^{src} denotes the incident flux from the source. n_1 is the first component of the normal vector $\vec{n}$ as defined in Section 3.3.1. If the etch yield is independent of the incidence direction, the surface velocity can simply be set proportional to the total flux. Therefore the surface velocity can be written as

$$V(\vec{p}) = -V_{\text{max}} \cdot \max(0, -n_1(\vec{p})), \tag{4.31}$$

where V_{max} is the maximum etch rate for normal incidence. As a demonstration, directional etching is applied to the structure given in Figure 4.12. The result, after a process time of 60 time units, is shown in Figure 4.14. The maximum etch rate was assumed to depend on the material. $V_{\text{max}} = 0.025$ and $V_{\text{max}} = 1$ (grid spacings per time unit) were assumed for the mask ($\mathcal{M}_3$) and the substrate ($\mathcal{M}_1$), respectively. The average computation times for time integration, normal calculation, and directional visibility check are given in Table 4.2 and prove the linear scaling with surface size.

The rounding at the bottom observed in Figure 4.14 can be ascribed to the finite difference scheme for solving the LS equation. All schemes inherently introduce some amount of dissipation in order to obtain a stable solution. The rounding can be reduced, if a finer grid with a smaller grid spacing is used for the calculation.

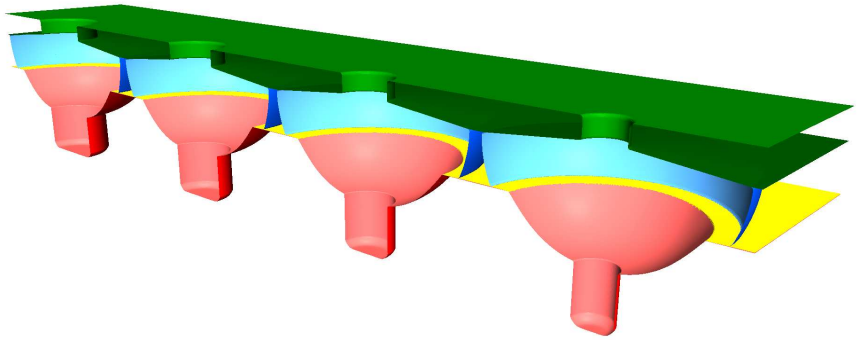


Figure 4.14: The structure given in Figure 4.12 after processing with a directional material-dependent etch process.

Scale factor	0.5	1.0	1.5	2.0	2.5
Lateral grid resolution	600×100	1200×200	1800×300	2400×400	3000×500
Time integration (avg.)	0.46 s	2.00 s	4.16 s	7.42 s	12.02 s
Visibility test (avg.)	0.02 s	0.08 s	0.17 s	0.30 s	0.47 s
Normal calc. (avg.)	0.08 s	0.32 s	0.71 s	1.28 s	2.04 s
Num. time steps	302	602	903	1203	1505

Table 4.2: The average computation times for a time integration step, the directional visibility test, and the normal vector calculation for the directional etching process shown in Figure 4.14.

4.7.2 Simple Bosch Process Simulation

The multi-LS technique, in combination with directional etching, enables the simulation of a Bosch process using a simplified model. The Bosch process is used for high aspect ratio etching by alternating passivation and etching cycles [67]. The deposition of a passivation layer protects the side walls from chemical etching during the subsequent etching cycle. Directional etching caused by ion bombardment removes the passivation layer at the bottom, so that the radicals are able to attack the substrate. The basic principle of a Bosch process is shown in Figure 4.15.

In a simplified model chemical etching and deposition can be assumed to be isotropic. Physical etching due to ions is regarded as perfect directional. Figure 4.16 shows the development of a 2.5 µm-hole during a Bosch process. The deposition rate has been set to $10\,\mathrm{nm\,s^{-1}}$. The chemical etch rate has been $3\,\mathrm{nm\,s^{-1}}$, $90\,\mathrm{nm\,s^{-1}}$, and $4.5\,\mathrm{nm\,s^{-1}}$ for the passivation layer, the mask, and the substrate, respectively. $5.47\,\mathrm{nm\,s^{-1}}$, $11.8\,\mathrm{nm\,s^{-1}}$, and $59.1\,\mathrm{nm\,s^{-1}}$ have been the corresponding directional etch rates. 20 cycles with 5 s deposition and 12 s etching were computed. The grid spacing has been set to 25 nm which corresponds to lateral grid extensions of 140×70. For the time evolution $C_{\mathrm{CFL}} = 0.5$ has been used.

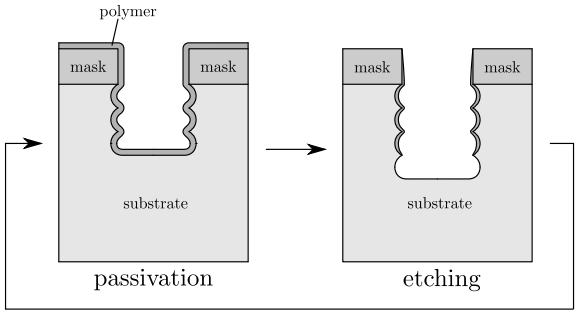
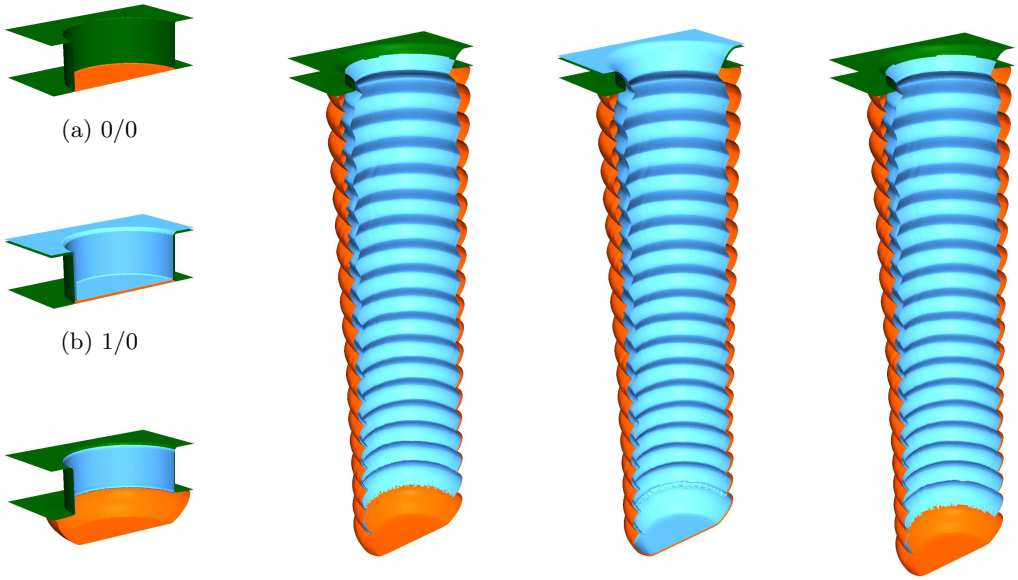


Figure 4.15: A schematic illustration of the Bosch process. The deposition of a passivation layer protects the sidewalls during the subsequent etching cycle.



(a) 0/0 (b) 1/0 (c) 1/1 (d) 19/19 (e) 20/19 (f) 20/20

Figure 4.16: The simulation of a Bosch process. The corresponding level set representation is shown for different times. The corresponding number of applied deposition and etching cycles can be retrieved from the subfigure captions.

The Bosch process is also an example which requires an accurate description of the thin passivation layer. Furthermore, the accurate resolution of the material-dependent surface velocities over time reduces the accumulation of large errors. Since all applied methods including the time evolution and the visibility check can be performed in linear time, the calculation of this simplified Bosch process is very fast. The computation time on an Intel Core 2 Duo E6600 processor clocking at 2.4 GHz is approximately 10 min.

4.8 Void Detection

As described in Section 2.3.3 the particle transport can be neglected for some models, if the surface reaction is not limited by the amount of reactants arriving at the surface. However, it is necessary to check if a surface point is connected to the source, because otherwise no reaction would occur on the surface. Obviously, the surface of voids within a structure is not connected to the outside, and hence the surface velocity must be set equal to zero. Therefore, it is necessary to find all existing voids. Since voids can be produced during deposition or disappear during the etching processes, the geometry must be tested for voids at every time step.

One way to check the geometry for voids is to extract an explicit surface from the implicit surface representation using a technique such as the marching cubes algorithm [70] and to check the triangulated version of the surface for connectivity. However, since this method is computationally very expensive, it would be convenient to have an efficient method to detect voids directly using the implicit LS surface representation.

4.8.1 Connected Components

The LS function which represents the surface $\mathcal{S}$, partitions the simulation domain into connected components. In the following section a fast algorithm is presented which uses inherent properties of the H-RLE data structure to determine the corresponding component each grid point belongs to. The determination of connected components gives information about existing voids. If there are no voids, there are only two components which correspond to the bulk material and the region above the surface as part of the process chamber. The obtained connectivity information can also be used to ensure that the geometry of voids does not change after they have been formed.

Two neighboring grid points are defined to be connected, if and only if they have the same LS sign. If they do not have the same sign, they are separated by the zero LS which is the surface. The connectivity relations between neighboring grid points can be described by a graph, where each grid point corresponds to a vertex. The connectivity of two neighboring grid points is represented by an edge between the corresponding vertices. According to elementary graph theory the connected components of a graph can be determined with a complexity of $\mathcal{O}(N_\mathrm{V} + N_\mathrm{E})$, where N_V and N_E denote the number of vertices and edges, respectively [117]. Obviously, setting up a full graph with vertices for each point on the regular grid is not reasonable, since the memory requirements and the computation of the connected components scale with the domain size and not linearly with the surface size.

4.8.2 Graph Setup Algorithm

If the H-RLE data structure is used, the utilization of the following properties allows the setup of a reduced graph which already combines several grid points within a vertex, and consequently, for which it is much easier to determine its connected components:

- As already mentioned in Section 3.5.4 the H-RLE data structure leads to a segmentation of the grid. Such a segment is either a defined grid point or an undefined run which combines one or more undefined grid points with the same LS sign (compare Figure 3.7).
- All grid points within a segment are connected. The connectivity follows for undefined runs from the fact that all contained grid points are neighbored and have the same sign. Hence, if any points of two different neighboring segments are connected, all of their points are also connected to each other.
- Two segments are neighbored, if and only if at least one of their corresponding first points is a neighbor to the other segment. The first point of a segment means the first point according to the lexicographical order given by the HRLE data structure. Its index vector is equal to the start index vector of a basic iterator positioned at the corresponding segment. As a consequence, it is sufficient to obtain all required connectivity relations between segments, by testing the neighboring points of all first points for connectivity.

To set up the reduced graph an array is needed to store a reference to the corresponding vertex for each segment in the H-RLE data structure. Using a basic iterator the H-RLE data structure is sequentially traversed and for each segment the following two tasks are performed:

1. The neighboring points of the first grid point in the current segment are tested for connectivity. If none of the corresponding connected neighbor segments is assigned to a vertex, a new vertex is inserted into the graph to which the current segment is assigned. Otherwise, the current segment is assigned to an arbitrary vertex to which a connected neighbor belongs.
2. All connected neighbor segments which do not belong to any vertex are assigned to the same vertex as the current segment. If there is a connected neighbor belonging to a different vertex, a new edge between the corresponding vertices is inserted in the graph.

Figure 4.17 shows an example with a LS representing a geometry with a void. After the procedure each segment is assigned to a vertex of the reduced graph. Due to the incorporation of connectivity relations during the setup, the number of vertices of the reduced graph is usually only a fraction of the number of defined grid points.

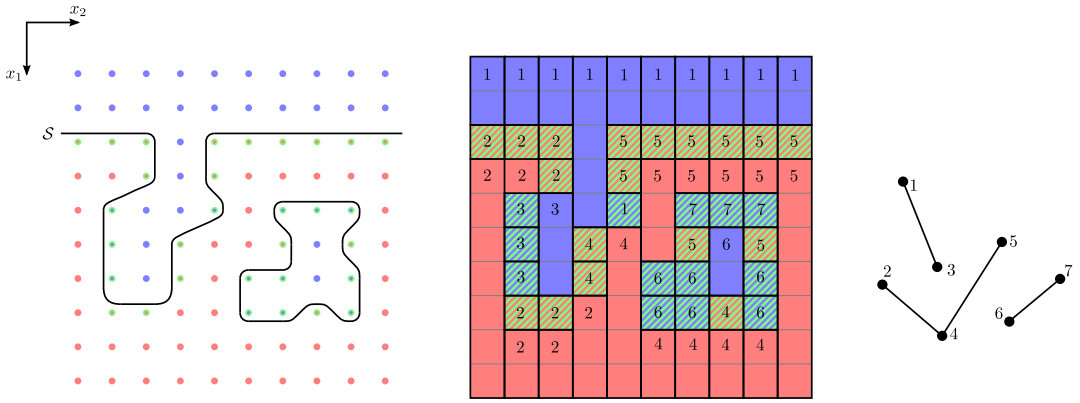


Figure 4.17: (a) The surface $\mathcal{S}$ of a geometry with a void. Blue and red points have positive and negative LS values, respectively. Defined grid points are also colored green. (b) The corresponding segments of the H-RLE data structure. Their numbers give the vertex of the reduced graph they are assigned to. (c) The reduced graph which is set up to find the connected components.

4.8.3 Algorithmic Complexity

The setup of the graph requires a sequential traversal over the H-RLE data structure. A first order finite difference stencil is used to enable fast access to neighbor grid points. Hence, the neighbor grid points of the first point of each segment can be tested for connectivity in constant time. As a consequence, the overall complexity required to set up the reduced graph exhibits a linear scaling with the number of defined grid points N_D. For the size of the reduced graph $N_\mathrm{E} + N_\mathrm{V} \leq \mathcal{O}(N_\mathrm{D})$ holds, since each segment in the H-RLE data structure leads to the insertion of, at most, one vertex and $2D$ edges, if D is the number of dimensions. As previously mentioned, the connected components of a graph can be obtained with linear complexity, which leads to an overall algorithmic complexity of $\mathcal{O}(N_\mathrm{D})$.

The memory requirements are also optimal. For each segment of the H-RLE data structure a reference of the corresponding vertex must be stored. The memory requirements for the reduced graph can usually be neglected, because, in practice, the number of vertices is much smaller than the number of segments.

4.8.4 Preservation of Voids

The connectivity information can be used to ensure that voids do not change over time. If the LS values of all active grid points which do not belong to and are not connected to any neighbor grid point belonging to the region above the surface, are not changed during time integration, the shapes of all voids are maintained. Depending

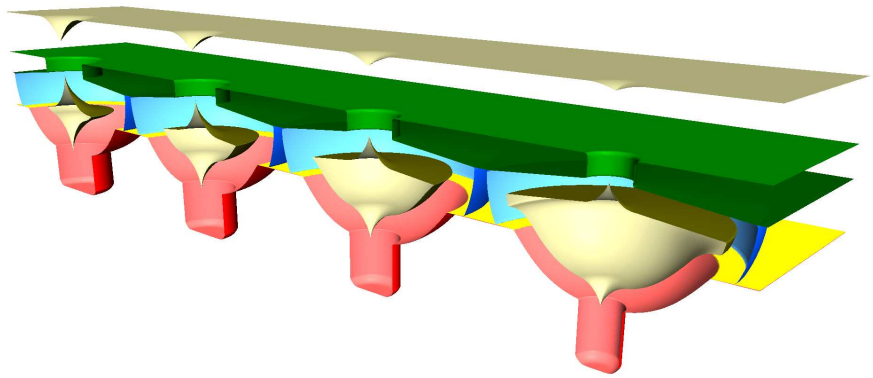


Figure 4.18: Isotropic deposition of a 60 grid spacings thick layer onto the structure given in Figure 4.14. Due to the varying hole diameters, the voids form at different points of time leading to different thicknesses of the deposited layer within the cavities.

on the orientation of the surface, the region above the surface is represented by the connected component which contains the first or the last grid point in the H-RLE data structure.

4.8.5 Isotropic Deposition

The presented void detection algorithm can be applied to isotropic deposition in order to ensure that voids do not change from the point in time they are formed. At every time step the void detection algorithm is used to determine all active grid points which must not change in order to preserve the shape of voids. The LS values of all other active grid points are updated in time. Figure 4.18 shows an example, where the void detection algorithm has been applied during a deposition process. Since the hole diameters are not equal, the cavities are disconnected from the source at different times.

Scale factor	0.5	1.0	1.5	2.0	2.5
Lateral grid resolution	600×100	1200×200	1800×300	2400×400	3000×500
Time integration (avg.)	0.48 s	2.52 s	4.40 s	7.73 s	12.42 s
Void detection (avg.)	0.18 s	0.73 s	1.65 s	2.93 s	4.68 s
Num. of vertices (max.)	87	157	309	429	615
Num. time steps	563	1188	1690	2373	2818

Table 4.3: The average computation time for a time integration step and the void detection algorithm. The number of vertices of the maximum reduced graph during the entire simulation is also given, and is very small compared to the number of defined grid points.

Table 4.3 lists the average calculation times for time integration and void detection for the isotropic deposition process. Furthermore, the number of vertices of the maximum reduced graph during the simulation is also given. Obviously, the number of vertices is very small compared to the number of defined grid points, which is at least in the order of the lateral grid extensions. Hence, the memory consumption of the reduced graph can be neglected, as already stated previously.

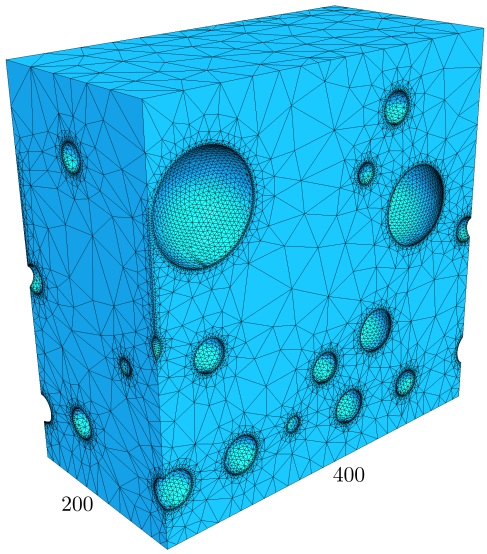
4.8.6 Isotropic Etching

Another application of the void detection algorithm is perfect isotropic etching applied to structures with voids. The surface is only advanced at locations, where it is connected to the source. As an example, isotropic etching of a "Swiss cheese"-like structure has been simulated. Figure 4.19 shows the initial geometry and the evolution over time. The etch rate is assumed to be 1 grid spacing per time unit.

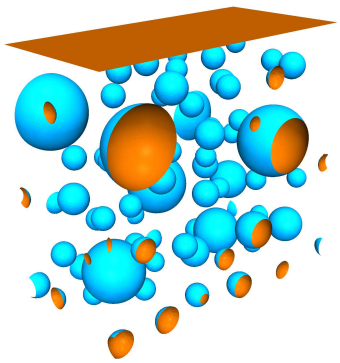
4.9 Surface Extraction

For visualization or further processing an explicit surface representation is more convenient than the implicit LS representation. To obtain a line segmentation or a triangulation from the LS function, the marching squares or the marching cubes algorithm [70] can be applied in two or three dimensions, respectively. The grid cells are processed in sequential order. If all corner grid points have the same LS sign, the surface does not intersect the cell. If there are any grid points with different signs, a surface segmentation is set up for the corresponding cell. The surface vertices are obtained as the intersection points of the zero LS with the cell edges. The surface elements of a cell are obtained using a predefined lookup table from the LS signs of all its corners. Since all grid cells need to be processed, the original algorithm scales with the domain size.

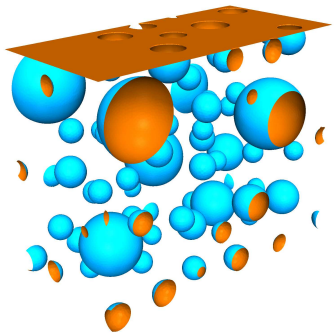
However, if the H-RLE data structure is used to represent the LS function, this algorithm can be implemented with linear complexity with respect to the number of defined grid points, which is usually proportional to the surface area. A cell-shaped stencil of 2^D iterators is used and moved across the H-RLE data structure. These iterators enable easy access to the LS values at all corners, as needed to determine any intersection points on edges. However, references to all new vertices on edges, which are attached to cells that have not been processed yet, are buffered in queues. When the stencil of iterators reaches a neighboring cell, according to the lexicographical processing order, vertices on edges, which have already been computed can easily be found. This avoids adding the same vertex twice and ensures the connectivity of all surface elements throughout neighboring grid cells.



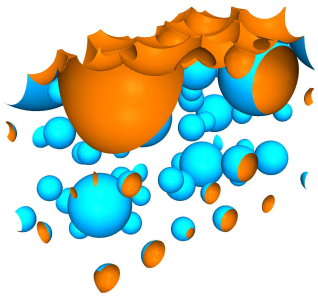
(a) Initial geometry with lateral extensions 200×400.



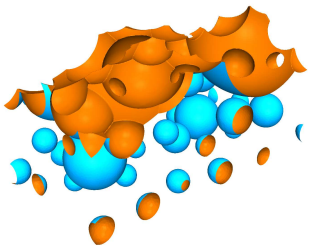
(b) $t = 0$



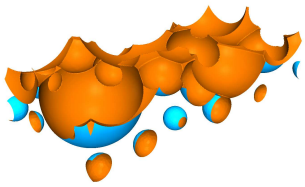
(c) $t = 40$



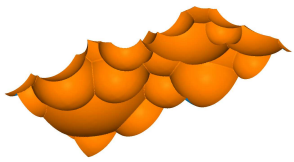
(d) $t = 80$



(e) $t = 120$



(f) $t = 160$



(g) $t = 200$

Figure 4.19: Isotropic etching of a "Swiss cheese"-structure with a constant etch rate of 1 grid spacing per time unit. (a) Initial geometry. (b)-(g) Illustration of the time evolution of the corresponding zero LS.

4.10 Parallelization

The increasing number of cores which are integrated in a single processor requires algorithms that are able to run in parallel. A parallelized version of the sparse field LS method for shared memory machines is already presented in [11]. There, a full grid is used which is distributed over multiple threads by partitioning the grid into slabs. If the H-RLE data structure is used, parallelization is more complicated. Run-length encoding is not predestined for parallelization, since the data structure is set up serially. In the following section a technique is described which distributes the information over multiple H-RLE data structures to enable parallel processing.

4.10.1 Parallelization Strategy

In order to achieve a high efficiency, all computational work should be uniformly distributed over all CPUs. According to an idea described in [11] a good load balance is obtained for the sparse field LS method, if the number of processed active grid points is the same for all CPUs. This can be achieved by analyzing the distribution of active grid points along a certain grid direction and defining slabs of the grid which all contain approximately an equal number of active grid points. If each thread processes one of these slabs, the expected runtimes are similar, and a good parallel efficiency can be obtained.

If a non-static data structure, such as the H-RLE data structure, is used instead of a full grid, parallelization is more complicated. As described previously in Section 4.3 the data structure must be rebuilt several times for every time step. Adding grid points to the data structure is only possible sequentially and in lexicographical order, and hence cannot be accomplished in parallel. A solution of this problem is to use as many H-RLE data structures as CPUs. This way, each thread may add grid points to its own H-RLE data structure at the same time. However, it must be ensured that all grid points are processed by exactly one thread. Therefore, the entire grid needs to be partitioned in advance. In order to maintain all the good properties of the H-RLE data structure, such as the fast sequential access and the small memory requirements, it is advantageous to divide the grid into sequences of grid points which contain approximately an equal number of active or defined grid points.

If N_{CPU} CPUs are used, the partitioning is characterized by N_{CPU} index vectors. Each index vector represents the start of a sequence of consecutive grid points, which is assigned to a thread and which is stored in the corresponding H-RLE data structure. For grid points stored in the H-RLE data structure of another thread, new run codes are introduced. For each CPU a new run code is defined, which provides information which H-RLE data structure houses the corresponding grid points. Figure 4.20 demonstrates the parallelization of the example given in Figure 3.6 and Figure 3.7. There, the LS values and undefined runs are distributed over four separate H-RLE data structures.

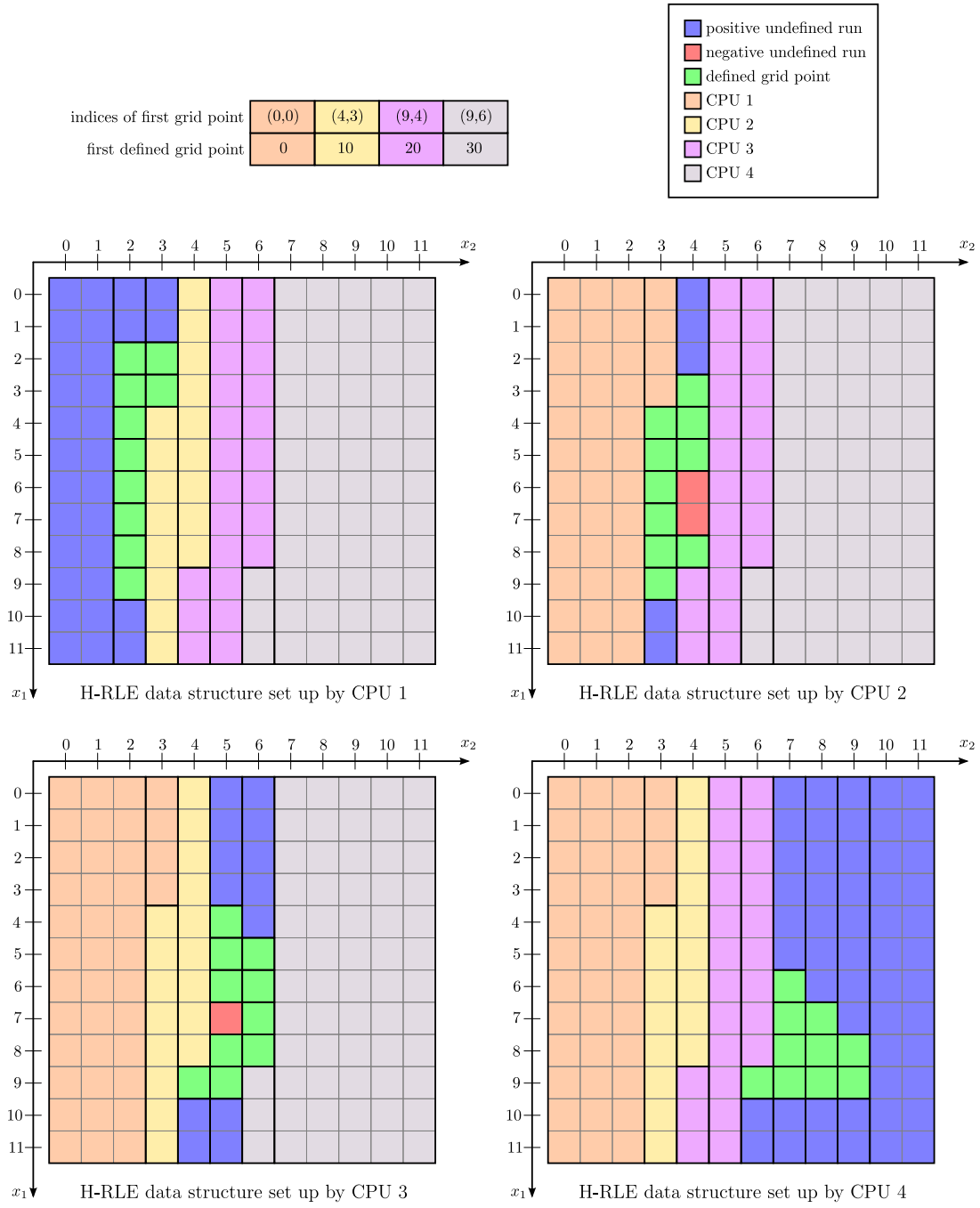


Figure 4.20: The parallel version of the H-RLE data structure given in Figure 3.7. An array of index vectors defines the grid segmentation. Each CPU processes the grid points of one segment in lexicographical order and writes the updated LS values into an own H-RLE data structure. For all other grid points which do not belong to the current CPU, run codes are inserted instead. They describe in which H-RLE data structure these points are stored.

4.10.2 Data Access

On shared memory machines the access to the H-RLE data structures of other threads is not difficult. Random access is realized by first determining the H-RLE data structure which contains the information of the corresponding grid point. This requires a search within the index vector array which defines the grid segmentation. This increases the complexity of a random access to $\mathcal{O}(\log N_{\mathrm{CPU}} + \log N_{\mathrm{D}})$ in the worst case.

Similarly, sequential access does not require any major changes. The iterators described in Section 4.2 can be reused with a small modification. If an iterator reaches a run code annotating that the corresponding grid points can be found in another H-RLE data structure, a random access operation is performed on that data structure. However, on average, sequential access can still be performed in constant time. This guarantees a linear complexity of the sparse field LS method for the parallel data structure.

The lexicographical order of defined grid points within the (non-parallel) H-RLE data structure leads implicitly to a numbering. Each defined grid point can be unambiguously identified by the array index of the corresponding LS value. This number is very useful when assigning additional data to defined grid points. To obtain this identification number from the parallel H-RLE data structure, an offset must be added to the corresponding array index. For this purpose an additional array is introduced to store the index of the first defined grid point, as shown in Figure 4.20.

4.10.3 Benchmarks

To test the parallel efficiency, the surface evolution was calculated for a sphere expanding at constant speed. The calculation was performed using 1, 2, 4, 8, and 16 cores of AMD Opteron 8435 processors clocking at 2.6 GHz. The corresponding average

	$d = 100$		$d = 1000$		$d = 10\,000$	
Num. CPUs	Time	Efficiency	Time	Efficiency	Time	Efficiency
1	60.8 ms	100.0 %	6.19 s	100.0 %	636 s	100.0 %
2	32.3 ms	94.2 %	3.18 s	97.4 %	331 s	96.0 %
4	18.8 ms	80.7 %	1.78 s	87.0 %	179 s	89.0 %
8	11.6 ms	65.6 %	0.94 s	81.9 %	95 s	83.4 %
16	6.9 ms	55.0 %	0.49 s	78.2 %	51 s	78.5 %

Table 4.4: Benchmarks for a time integration step of a sphere expanding with constant speed. The computation times as well as the parallel efficiency are given for varying sphere diameters d and number of used CPUs.

calculation times for a single time step and for different sphere diameters d (measured in grid spacings) are listed together with the parallel efficiency in Table 4.4.

According to Amdahl's law [9] the parallel efficiency decreases with the number of CPUs due to sequentially processed parts of the program. In case of 16 cores a parallel efficiency of approximately 78 % could be achieved except for the smallest sphere diameter $d = 100$. For smaller structures the overhead due to thread synchronization is more relevant, which results in a worse efficiency. Table 4.4 also shows the good scalability with surface size. If the diameter is multiplied by 10, the surface of the sphere is increased by a factor of 100, which is well reproduced by the listed runtimes.

5 Surface Rate Calculation

The simulation of surface evolution requires the calculation of the surface velocity at every time step. For advanced models, the surface velocity depends on the particle transport. According to the surface kinetics model described in Section 2.3, the surface velocity V is assumed to be a function of a certain number of surface rates $R_1, \ldots, R_{N_\mathrm{R}}$ (2.25). This chapter is devoted to the calculation of these surface rates under the assumption of ballistic particle transport within the feature-scale region, as described by (2.14). Particles entering the feature-scale region through the source plane $\mathcal{P}$ follow the given arriving flux distribution Γ_src. Particle reemission, modelled by the reemission probability function G (2.15), must also be incorporated. In summary the problem states as follows:

- Given: $\mathcal{S}$, Γ_src for all $\vec{x} \in \mathcal{P}$, and G for all $\vec{x} \in \mathcal{S}$.
- Required: $R_1, \ldots, R_{N_\mathrm{R}}$ (in order to calculate V) for all $\vec{x} \in \mathcal{S}$.

Two completely different approaches for solving this problem are presented in the following sections.

5.1 Conventional Approach

One way to calculate the surface rates is by direct integration. This requires an appropriate discretization of the domain of the flux distribution function $\Gamma = \Gamma(\vec{x}; q, \vec{\omega}, E)$. Hence, the flux distribution at a certain surface point $\vec{x}$ is approximated as a superposition of a finite number of appropriate basis functions $b_1, b_2, \ldots, b_{N_\Gamma}$

$$\Gamma(\vec{x}; q, \vec{\omega}, E) \approx \sum_{l=1}^{N_\Gamma} B_l(\vec{x}) b_l(\vec{x}; q, \vec{\omega}, E). \tag{5.1}$$

$B_l(\vec{x})$ are the corresponding coefficients. If the basis functions represent an orthonormal basis, thus

$$\sum_{q=1}^{Q} \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} b_l(\vec{x}; q, \vec{\omega}, E) b_k(\vec{x}; q, \vec{\omega}, E)\, d\Omega\, dE = \delta_{lk}, \tag{5.2}$$

the coefficients $B_l(\vec{x})$ can be calculated using

$$B_l(\vec{x}) = \sum_{q=1}^{Q} \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} b_l(\vec{x};q,\vec{\omega},E)\Gamma(\vec{x};q,\vec{\omega},E)\,d\Omega\,dE. \tag{5.3}$$

Insertion of the ballistic transport equation (2.14) yields

$$B_l(\vec{x}) = S_l(\vec{x}) + \int_{\mathcal{S}} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \sum_{q=1}^{Q} \int_0^\infty b_l(\vec{x};q,\vec{\omega},E)\Gamma_{\mathrm{re}}(\vec{x}';q,\vec{\omega},E)\,dE\,dA', \tag{5.4}$$

where S_l denotes the source term

$$S_l(\vec{x}) := \int_{\mathcal{P}} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \sum_{q=1}^{Q} \int_0^\infty b_l(\vec{x};q,\vec{\omega},E)\Gamma_{\mathrm{src}}(\vec{x}';q,\vec{\omega},E)\,dE\,dA'. \tag{5.5}$$

Here, $\mathrm{vis}(\vec{x},\vec{x}')$ is the visibility function which returns 1 or 0, if the surface points $\vec{x}$ and $\vec{x}'$ are in line or sight or not, respectively. Utilizing expression (2.15) finally gives

$$B_l(\vec{x}) = S_l(\vec{x}) + \int_{\mathcal{S}} \sum_{k=1}^{N_\Gamma} T_{lk}(\vec{x},\vec{x}')B_k(\vec{x}')\,dA' \tag{5.6}$$

with

$$\begin{aligned} T_{lk}(\vec{x},\vec{x}') := \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \int_0^\infty \sum_{q=1}^{Q} b_l(\vec{x};q,\vec{\omega},E)\cdot \\ \cdot \sum_{q'=1}^{Q} \int_0^\infty \int_{\vec{\omega}'\cdot\vec{n}(\vec{x}')<0} G(\vec{x}';q,\vec{\omega},E;q',\vec{\omega}',E')b_k(\vec{x}';q',\vec{\omega}',E')\,d\Omega'\,dE'\,dE. \end{aligned} \tag{5.7}$$

To solve the surface integral equation (5.6) numerically, a discretization of the surface is needed. Triangle [68] or voxel elements [6] are conventionally used in three dimensions. If N_{S} denotes the number of discretization elements of the surface, the number of unknowns is equal to $N_{\mathrm{S}} \cdot N_\Gamma$. If N_Γ is large, thus if many basis functions are used to resolve the direction and energy dependence of the flux distribution, the number of variables, which must be kept in memory, is very large. This is especially a problem in three dimensions, where N_{S} is already very large by nature. Therefore, only a very small number of basis functions is commonly used in practice. Very often, only one basis function is used per particle type

$$b_l(\vec{x};q,\vec{\omega},E) := \delta_{lq}, \tag{5.8}$$

which reduces the number of unknowns to $N_\mathrm{S} \cdot Q$. As a consequence, the corresponding coefficients are equal to the respective total arriving flux of particles of species q

$$B_l(\vec{x}) = \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})<0} \Gamma(\vec{x}; q, \vec{\omega}, E)\, d\Omega\, dE = F_q(\vec{x}). \tag{5.9}$$

The particle transport equation (2.14) simplifies to

$$\begin{aligned} F_q(\vec{x}) = &\int_\mathcal{P} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \int_0^\infty \Gamma_\mathrm{src}(\vec{x}'; q, \vec{\omega}, E)\, dE\, dA' + \\ &\int_\mathcal{S} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \int_0^\infty \Gamma_\mathrm{re}(\vec{x}'; q, \vec{\omega}, E)\, dE\, dA', \end{aligned} \tag{5.10}$$

and the reemitted flux distribution (2.15) can be written as

$$\Gamma_\mathrm{re}(\vec{x}'; q, \vec{\omega}, E) = \sum_{q'=1}^{Q} F_{q'}(\vec{x}') \int_{\vec{\omega}'\cdot\vec{n}(\vec{x}')<0} \int_0^\infty G(\vec{x}'; q, \vec{\omega}, E; q', \vec{\omega}', E')\, dE'\, d\Omega'. \tag{5.11}$$

Here, the incident direction and the energy do not play a role in the reemitted flux distribution Γ_re. The choice of the basis according to (5.8) assumes implicitly that the direction and the energy of reemitted particles is independent of the incident direction and energy. Therefore, effects such as specular-like reflexions of ions (see Section 2.2.3) cannot be accurately described, except for unidirectional and monoenergetic ion sources, for which the reemission probability function is the same for all incident ions.

Similarly, if the chosen basis is not able to reproduce the direction or the energy dependence of the arriving flux distribution Γ, the surface rates cannot be computed properly, given that they depend on the incident angle or energy. Hence, non-trivial surface rates, such as sputter rates, cannot be calculated correctly using the simple basis (5.8) and inserting (5.1) into (2.24). Only for unidirectional and monoenergetic particle incidence the sputter rates can be calculated correctly.

At least the contribution of primary particles, which come directly from the source plane $\mathcal{P}$, to the surface rates can be calculated correctly with reasonable effort using the simple basis (5.8). Once the coefficients $B_l(\vec{x})$ are calculated, the reemitted flux distribution can be approximated by inserting (5.1) into (2.15). Then the surface rates can be calculated using the formula

$$\begin{aligned} R_l(\vec{x}) = &\int_\mathcal{P} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \sum_{q=1}^{Q} \int_0^\infty r_l(\vec{x}; q, \vec{\omega}, E)\Gamma_\mathrm{src}(\vec{x}'; q, \vec{\omega}, E)\, dE\, dA' + \\ &\int_\mathcal{S} \frac{-\vec{\omega}\cdot\vec{n}(\vec{x})}{\|\vec{x}-\vec{x}'\|^2}\,\mathrm{vis}(\vec{x},\vec{x}') \sum_{q=1}^{Q} \int_0^\infty r_l(\vec{x}; q, \vec{\omega}, E)\Gamma_\mathrm{re}(\vec{x}'; q, \vec{\omega}, E)\, dE\, dA' \end{aligned} \tag{5.12}$$

which is obtained by combining (2.14) and (2.24).

5.1.1 Algorithmic Complexity

First the computational effort is considered, which is necessary for setting up the system matrix of (5.6). In principle, every pair of surface elements must be tested, if reemitted particles from one element can reach the other. A reasonable lower bound estimation for the complexity of such a visibility test is $\mathcal{O}(\log N_\mathrm{S})$. Hence, the total complexity for the setup of the system matrix is at least $\mathcal{O}({N_\mathrm{S}}^2 \log N_\mathrm{S})$.

Solving the system of linear equations also exhibits a bad scaling behavior. The expected number of non-zero entries is $\mathcal{O}({N_\mathrm{S}}^2)$ [41]. A very efficient approach to solve the system is an iterative technique described in [6]. In this way usually only a small number of matrix-vector multiplications is necessary to obtain a convergent solution. Since the complexity of a matrix-vector multiplication is $\mathcal{O}(N_\Gamma {N_\mathrm{S}}^2)$, a quadratic scaling with surface size also governs the solution of the system.

5.1.2 Limitations

Due to the bad scaling behavior, the conventional approach is limited to small problem sizes. For large three-dimensional problems, where the number of surface discretization elements easily reaches several million, the calculation of the particle transport is very time and memory intensive without further simplifications.

One possible way to reduce the computational effort is the choice of a coarser discretization at regions with low curvature, as proposed in [41]. However, this approach does not only reduce the number of surface elements, it also reduces the spatial resolution of the flux distribution. This is a problem, since even on plane regions of the surface the flux can change abruptly due to shadowing. Hence, the size of the discretization elements should be in the order of the grid spacing used for the LS method.

Another drawback is the non-trivial and error-prone calculation of the matrix elements. Especially for more complicated basis functions, the evaluation of (5.7) is cumbersome. In [19] a simpler technique to calculate the system matrix elements based on a MC approach was investigated; however, this has not been proven to be very practical.

5.2 Ray Tracing

Another MC technique, which can be used to calculate the surface rates directly, is ray tracing [10, 114]. Ray tracing is a widely applied technique in computer graphics for rendering a scene. A large number of light rays is used to obtain a realistic picture. In

recent years, this MC technique has been successfully applied to topography simulation to calculate surface rates [8, 64, 95]. Especially if the particle trajectories are linear, which is the case for ballistic transport, the same algorithms and techniques can be applied in an analogous manner [A13, A16].

In order to simulate the particle transport and to obtain the surface rates, many particles are launched from the source plane $\mathcal{P}$, and their trajectories are calculated. Whenever a particle reaches the surface $\mathcal{S}$, it contributes directly to the local surface rates introduced in (2.24). For each surface point $\vec{x}$, for which the rates should be calculated, a surrounding neighborhood is defined. If the area of that neighborhood, denoted by $A_{\mathrm{ref}}(\vec{x})$, is small enough and the surface is smooth, the neighborhood can be assumed to be plane and orthogonal to the surface normal $\vec{n}(\vec{x})$. These neighborhoods of known area are necessary to relate the contribution of a single particle to the rates at point $\vec{x}$. An incident particle of type q_{inc} with direction $\vec{\omega}_{\mathrm{inc}}$ and energy E_{inc} striking the neighborhood of point $\vec{x}$ adds

$$\Delta R_l(\vec{x}) = \frac{F^{\mathrm{tot}} \cdot r_l(\vec{x}; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})}{N_{\mathrm{P}} \cdot A_{\mathrm{ref}}(\vec{x})}$$

$$\text{with } F^{\mathrm{tot}} := \int_{\mathcal{P}} \sum_{q=1}^{Q} \int_{0}^{\infty} \int_{\vec{\omega}\cdot\vec{n}_{\mathcal{P}}>0} \Gamma_{\mathrm{src}}(\vec{x}; q, \vec{\omega}, E)\, d\Omega\, dE\, dA \quad (5.13)$$

to the corresponding rate function $R_l(\vec{x})$. Here N_{P} denotes the total number of simulated particles. The particles are launched from the source plane $\mathcal{P}$ and must obey the flux distribution Γ_{src} (2.2). F^{tot} is the total flux of particles through $\mathcal{P}$. In order to account for reemission, new particles must be launched after striking the surface. Reemitted particles need to obey the conditional probability given by the model dependent reemission probability function G. The probability $p_{\mathrm{re}}(q)$ that a particle of type q is reemitted is given by

$$p_{\mathrm{re}}(q) = \frac{\int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})>0} G(\vec{x}; q, \vec{\omega}, E; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})\, d\Omega\, dE}{\sum_{q'=1}^{Q} \int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})>0} G(\vec{x}; q', \vec{\omega}, E; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})\, d\Omega\, dE}. \quad (5.14)$$

This probability can take values larger than 1. For example, the average number of sputtered particles is often larger than 1. Hence, to satisfy statistics, multiple particles must be launched. However, this leads to a very large number of secondary particles which all must be simulated.

A better strategy is to introduce a volume or weight factor w assigned to a particle as proposed in [118] and which allows to control the number of reemitted particles. Initially, when a particle is launched from the source, its weight factor is set to unity ($w = 1$). An incident particle contributes to the local rates according to its current weight factor w_{inc}

$$\Delta R_l(\vec{x}) = \frac{w_{\mathrm{inc}} \cdot F^{\mathrm{tot}} \cdot r_l(\vec{x}; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})}{N_{\mathrm{P}} \cdot A_{\mathrm{ref}}(\vec{x})}. \quad (5.15)$$

Instead of reemitting several particles of type q according to the probability $p_{\mathrm{re}}(q)$, it is possible to initialize just one new particle with an adequate weight factor calculated as

$$w = w_{\mathrm{inc}} \cdot p_{\mathrm{re}}(q). \tag{5.16}$$

The weight factor can be used as termination criterion. If the weight factor is below a certain critical level, so that the contribution to the surface rates is negligible, the reemission of new particles can be avoided [A20].

The directional and energy distribution $f(\vec{\omega}, E)$ of a reemitted particle species q obeys

$$f(\vec{\omega}, E) = \frac{G(\vec{x}; q, \vec{\omega}, E; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})}{\int_0^\infty \int_{\vec{\omega}\cdot\vec{n}(\vec{x})>0} G(\vec{x}; q, \vec{\omega}, E; q_{\mathrm{inc}}, \vec{\omega}_{\mathrm{inc}}, E_{\mathrm{inc}})\, d\Omega\, dE}. \tag{5.17}$$

The accuracy of the statistically calculated rates $R(\vec{x})$ is primarily influenced by the average number of incident particles on a neighborhood, which is proportional to the total number of simulated particles N_{P} and the corresponding area $A_{\mathrm{ref}}(\vec{x})$. To improve the accuracy, either more particles must be simulated or the areas of the neighborhoods must be increased at the expense of the spatial resolution.

The main computational task of this ray tracing technique is to determine the first surface intersection of rays and to test for intersections with the neighborhoods of all surface points $\vec{x}$, for which the surface rates should be calculated. The choice of surface representation and of neighborhoods, as well as the corresponding intersection tests are discussed in the following sections.

5.2.1 Surface Representation

The most common way to represent the surface is a line segmentation in case of two dimensions [8], or a triangulation in case of three dimensions [64, 95]. The advantage of such explicit representations is that ray intersection tests are relatively simple for the surface elements (compare Appendix A). Furthermore, surface elements can be used as neighborhoods. For example, in [95] the particle flux is calculated for each triangle by counting the number of incident particles and dividing by the triangle area. However, if the surface is extracted using the marching cubes algorithm [70], the surface elements vary significantly in size. Hence, to obtain a good accuracy for all corresponding surface points, their neighborhoods must be extended over multiple surface elements, which requires an additional data structure and complicates the entire algorithm [A13].

Another drawback of the explicit surface representation is that it needs to be extracted from the implicit LS surface representation, which is needed for surface evolution using the LS method, at every time step. Hence, calculation time and memory are wasted due to this additional surface representation.

An improved strategy is to apply ray tracing directly on the implicit LS representation [112, A16]. To find the intersection of a ray with the surface, the LS function has to be bi- or trilinearly interpolated, depending on the number of dimensions D. Inserting the parameterization of the ray into the interpolation formula gives a quadratic or cubic polynomial, respectively. Then the first real root, if it exists, corresponds to the intersection point. It is important for a fast ray tracing algorithm, that the intersection test requires a minimum number of numerical operations. In Appendix B a detailed and efficient ray–isosurface intersection test is presented.

Multi-linear interpolation within a grid cell requires the LS values of all 2^D grid points. Hence, if the sparse field method is used, it must be ensured through dilation (see Section 4.3.3), that all these grid points are defined and their LS values are available.

5.2.2 Tangential Disks

The LS method requires a surface velocity field which is commonly obtained by extrapolating the velocities computed on the surface (compare Section 3.2.4). If the active grid points are very close to the surface, which is the case for the sparse field method, it is possible to calculate the surface rates directly for those grid points [112, A13]. In this way the extrapolation procedure can be avoided and no extra memory is needed to store the velocities for the discretized surface.

As proposed in [A16, A20] tangential disks $\mathcal{D}(\vec{p})$ can be defined for all points on the surface, which are closest to any active grid points $\vec{p} \in \mathcal{L}_0$. Hence, for each active grid point $\vec{p}$ a disk is defined which serves as a neighborhood for its corresponding closest surface point $\vec{x}_{\text{cp}}(\vec{p})$. The closest surface point $\vec{x}_{\text{cp}}(\vec{p})$, which is also the midpoint of the tangential disk $\mathcal{D}(\vec{p})$, is approximated using (3.23). The orientation of the disk is given by the surface normal which is approximated by $\vec{n}(\vec{p})$ as defined in (3.22). Hence, the tangential disk $\mathcal{D}(\vec{p})$ is described by

$$\mathcal{D}(\vec{p}) = \{\vec{x} : \|\vec{x} - \vec{x}_{\text{cp}}(\vec{p})\| \leq r \wedge (\vec{x} - \vec{x}_{\text{cp}}(\vec{p})) \cdot \vec{n}(\vec{p}) = 0\} \tag{5.18}$$

with r denoting the radius. Disks have the advantage that intersection tests with rays are relatively simple. A ray with parameterization $\vec{x}(t) = \vec{a} + t \cdot \vec{\omega}$ strikes the disk $\mathcal{D}(\vec{p})$ on the front side, if the following condition is fulfilled

$$\cos\theta > 0 \quad \wedge \quad \|\vec{x}_{\text{rel}} \cdot \cos\theta + \vec{\omega} \cdot (\vec{n}(\vec{p}) \cdot \vec{x}_{\text{rel}})\| \leq r \cdot \cos\theta \tag{5.19}$$

with $\cos\theta := -\vec{\omega} \cdot \vec{n}(\vec{p})$ and $\vec{x}_{\text{rel}} := \vec{a} - \vec{x}_{\text{cp}}(\vec{p})$. In this case the corresponding particle contributes to the rates of grid point $\vec{p}$ according to (5.15). To calculate the surface rates correctly for active grid points, whose corresponding tangential disks are partially behind the surface $\mathcal{S}$ due to the curvature, it is necessary to prolong the calculation of the particle trajectory for a few grid spacings from the surface intersection point (see Figure 5.1). However, the first intersection point with the surface is stored and used as a starting point for reemitted particles.

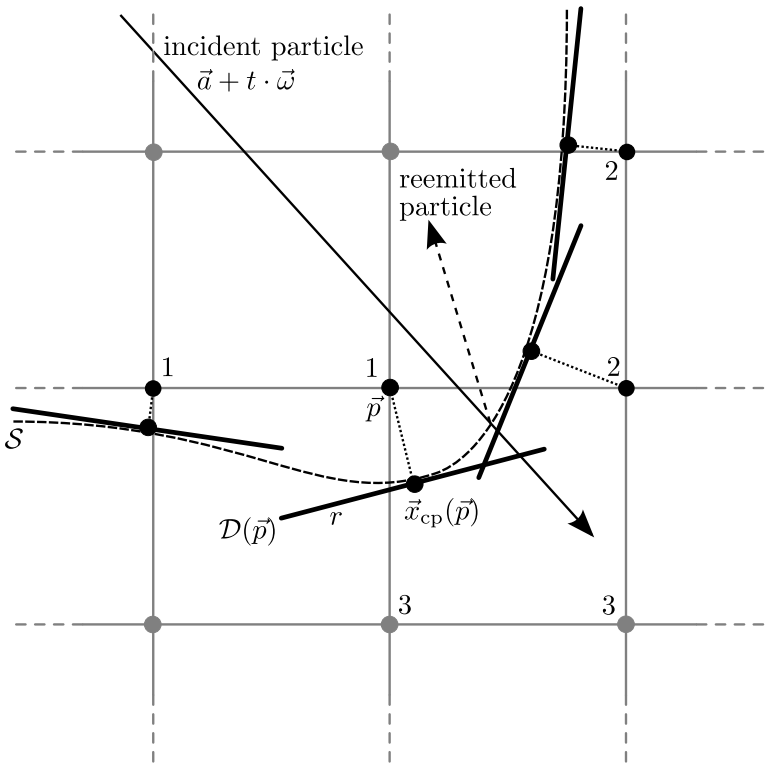


Figure 5.1: If a particle trajectory intersects any tangential disk (thick black) $\mathcal{D}(\vec{p})$, it contributes to the rates of the corresponding active grid point (black) $\vec{p} \in \mathcal{L}_0$. Due to the curvature of the surface $\mathcal{S}$, the calculation of the particle trajectory must be continued for a few grid spacings in order to obtain correct rates. However, new reemitted particles (dashed) are always launched from the surface intersection point. To avoid multiple intersection tests of the same disk, only those grid points which are opposite to the entry face are checked, if they are active and if their corresponding disk is intersected. The numbers show which points are processed in which cell.

The area of the disks is $A_{\mathrm{ref}} = \pi r^2$ or in case of two dimensions, where the disks correspond to tangential segments, $A_{\mathrm{ref}} = 2r$. Hence, the choice of the disk radius influences the statistical accuracy. Larger radii give better statistics at the expense of the spatial resolution which should be of same magnitude as the resolution of the surface. Hence the disk radius should be in the order of the grid spacing which was assumed to be unity.

In [A20] an upper limit for the radius was suggested. Within the sparse field method $|\Phi(\vec{p})| \leq \frac{1}{2}$ holds for all active grid points. Furthermore, the approximation for the gradient is almost always larger than 1, thus $\|\nabla\Phi(\vec{p})\| \geq 1$. This gives an upper limit for the distance to the approximated closest surface point $\|\vec{p} - \vec{x}_{\mathrm{cp}}(\vec{p})\| \leq \frac{1}{2}$ according

to (3.23). If the radius is chosen in such a manner that

$$r \leq \sqrt{1 - \left(\tfrac{1}{2}\right)^2} \approx 0.866, \tag{5.20}$$

the disk $\mathcal{D}(\vec{p})$ is within the 2^D grid cells which are adjacent to the active grid point $\vec{p}$. In very rare cases, when $\|\nabla\Phi(\vec{p})\| \geq 1$ is not fulfilled, it might be necessary to translate the disk towards $\vec{p}$, in order to fit it into the surrounding cells. Much smaller disk radii than the given upper limit lead to worse statistics without improving the spatial resolution. Therefore, it is recommended to use a disk radius of 0.8 grid spacings [A20].

This way the computational effort for ray–disk intersection tests is minimized. Within a grid cell only the 2^D corner grid points have to be checked, if they are active, and if their corresponding tangential disks $\mathcal{D}(\vec{p})$ are intersected. Hence, the same data structure can be used as the one needed for the multi-linear interpolation for the surface intersection test, which requires links to all its corners in order to access the corresponding LS values.

5.2.3 Particle Traversal

Particles are consecutively launched from the source plane $\mathcal{P}$ according to the arriving flux distribution Γ_{src}. The vertical position of $\mathcal{P}$ can be arbitrarily chosen, however, it must be above the surface. If the arriving flux distribution Γ_{src} varies spatially, it must be adapted correspondingly when shifting the source plane. For ray tracing a good choice is the lowest grid plane which is entirely above the surface. Hence, it is the first grid plane, where all grid points have a positive signed LS value. This plane limits the simulation domain on top. The simulation domain can also be limited at the bottom by the first grid plane which is entirely below the surface. Together with the lateral boundaries a rectangular simulation domain is obtained.

To find the first intersection of a particle trajectory with the surface all traversed grid cells need to be tested for an intersection within their interiors. If a surface intersection is found, new particles must be launched from the intersection point according to the reemission law. Their reemitted direction and energy as well as their particle species and their starting point are stored in a stack, because these particles will be processed later. First the trajectory of the incident particle hitting the surface is further calculated for a few grid spacings. This allows the proper calculation of the surface rates, even if the corresponding tangential disks are behind the surface (compare Figure 5.1). In this work the trajectories are prolonged for three grid spacings before discarding the particle and calculating the trajectory of the next one. The new particle is either obtained from the stack, if it is not empty, or else by generating a new particle launched from the source plane $\mathcal{P}$.

Continuous tests of all grid cell corners to check for active grid points, followed by the corresponding tangential disk intersection tests, lead to unnecessary computational

effort. The same grid point will likely be tested multiple times within different neighboring grid cells. A better strategy is to test only those grid points opposite to the face, through which the particle enters the grid cell (see Figure 5.1). In this way each grid point is only tested once.

5.2.4 Algorithmic Complexity

The algorithmic complexity of ray tracing is primarily given by the number of simulated particles which are required to obtain suitable accurate rates, and the effort for calculating a single particle trajectory. The number of simulated particles must scale with the surface area (measured in grid spacings) in order to keep the number of incidences on a disk constant. The surface area, in turn, can be said to be proportional to the number of grid cells which are intersected by the surface. These grid cells can be regarded as the surface discretization elements of the implicit surface representation. Let N_S be the number of these elements, which allows a comparison with the conventional approach. The number of simulated particles must be of order $\mathcal{O}(N_\mathrm{S})$.

The effort of tracing a single particle is given by the number of grid cells which must be traversed to find the first surface intersection. The expected number of traversed cells is $\mathcal{O}(\sqrt[D-1]{N_\mathrm{S}})$ [79]. As a consequence, the expected total computational costs scale with $\mathcal{O}(N_\mathrm{S}^{\frac{D}{D-1}})$. For three dimensions ($D = 3$) this is already an improved scaling behavior than that of the conventional approach.

The following sections describe how the expected computational effort for finding the first surface intersection point can be further reduced to $\mathcal{O}(\log N_\mathrm{S})$. As a consequence, the total effort is equal to $\mathcal{O}(N_\mathrm{S} \log N_\mathrm{S})$. Hence, for large structures with very large N_S, ray tracing is superior to the conventional approach for the calculation of the surface rates and does not require the simplifications of the general model.

5.2.5 Boundary Conditions

The simulation domain for ray tracing is limited in the vertical direction by the source plane $\mathcal{P}$ and the surface $\mathcal{S}$ (see Figure 2.1). The lateral directions are limited by boundary conditions. Periodic and reflective boundary conditions are most commonly used. Both types can easily be incorporated using ray tracing. If a particle would leave the feature-scale region at a periodic boundary, it reenters the simulation domain at the opposite boundary with same direction. In case of reflective boundaries the particle direction is reflected and the trajectory calculation is continued.

Periodic boundaries can only be applied, if the initial geometry is periodic, which means that opposite sides of the geometry fit together. By contrast, reflective boundaries can be applied regardless of the initial geometry. However, the application of reflective boundaries can be problematic for inclined particle incidence. Due to the

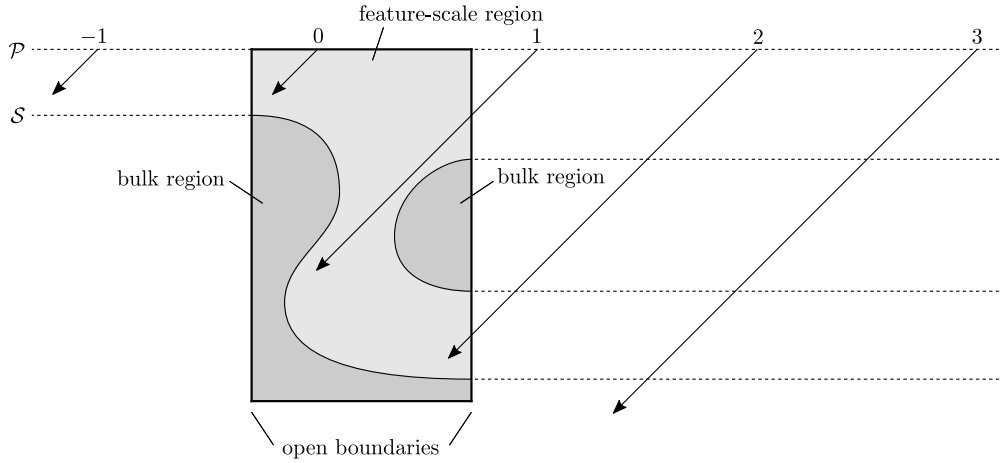


Figure 5.2: Additional particles are started from positions, which are offset by a multiple of the lateral domain extension, in order to account for the primary flux coming through the open domain boundaries.

reflection at boundaries, particles obtain a different direction than given initially by the arriving flux distribution Γ_{src}, which leads to an incorrect arriving flux distribution Γ on the surface. Only if the flux distribution Γ_{src} is symmetric with respect to the vertical grid direction, reflective boundaries can be applied. Hence, for all directions $\vec{\omega}$ and $\vec{\omega}'$ satisfying $(\vec{\omega} + \vec{\omega}') \times \vec{n}_{\mathcal{P}} = 0$, the corresponding values of the arriving flux distribution need to be equal, $\Gamma_{\text{src}}(\vec{x}; q, \vec{\omega}, E) = \Gamma_{\text{src}}(\vec{x}; q, \vec{\omega}', E)$.

Problems, for which Γ_{src} is not symmetric and the initial geometry is not periodic at the same time, neither reflective nor periodic boundaries can be applied. For such problems it is better to use open boundaries, where the geometry is thought to extend to infinity (see Figure 5.2). A particle leaving the simulation domain is not further treated. It is clear that this leads to wrong rates, especially for grid points close to the boundary. Since no particles enter the simulation domain through the boundary, the flux distribution is zero for the corresponding directions. To obtain correct results new particles must be launched from the boundaries. The calculation of the corresponding arriving flux distribution for points on the lateral boundaries is very difficult, because it can be composed of direct flux from the source plane and secondary flux due to reemission.

At least the direct flux can be fully incorporated, if the arriving flux distribution is equally distributed over the source plane $\mathcal{P}$. As usual, particles are randomly initiated from the source plane. However, additional particles of the same species are launched with same direction and energy from positions which are relatively offset by an integral multiple of the domain extension. The number of particle trajectories which must be computed can be limited from the outset, since only particles which intersect the simulation domain have to be considered. In the example shown in Figure 5.2 only the particles with numbers 0, 1, and 2 must be considered.

The entrance point into the simulation domain can easily be calculated for all particles which are started from the outside. However, it is necessary to check whether the simulation domain can be reached without intersecting the surface before entry. For example, particle 2 is not able to reach the surface within the simulation domain, because it would have to traverse the bulk region (which is extended to infinity). Therefore, each particle, starting from the outside, is initially traced along the boundary of the simulation domain, until it reaches the entrance point, in order to check, if it encounters any surface intersection.

If the arriving flux distribution Γ_{src} is not equally distributed over the source plane $\mathcal{P}$, but rather locally concentrated, as is the case for focused ion beam (FIB) applications, the choice of boundary condition does not affect the result, assuming the lateral extensions of the simulation domain are sufficiently large.

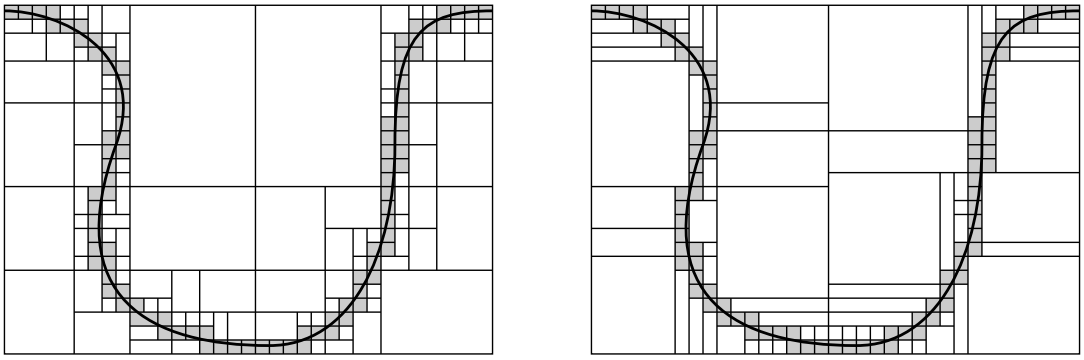
5.2.6 Spatial Subdivision

The cell-by-cell traversal to find intersections with the surface and the disks requires the determination of the face, through which a particle leaves a grid cell, in order to obtain the next cell. Intersections are only possible within grid cells which contain parts of the surface or parts of any disk. All other empty grid cells do not need to be tested for any kind of intersection. Therefore, it is possible to combine these empty grid cells to obtain larger boxes to be traversed simultaneously. In this way, if efficient data structures are used, which allow a fast determination of neighboring boxes, the computation time can be reduced. The goal is to find an appropriate decomposition of the simulation domain into a set of disjoint rectangular axis aligned subboxes $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{N_\text{B}}$:

$$\mathcal{B} = \bigcup_{i=1}^{N_\text{B}} \mathcal{B}_i \quad \text{and} \quad \mathcal{B}_i \cap \mathcal{B}_j \neq \emptyset \quad \text{for all} \quad i \neq j. \tag{5.21}$$

Here $\mathcal{B}$ denotes the bounding box of the simulation domain. Binary space partition is commonly used to obtain a suitable decomposition [10, 39]. Subsequent splitting of the domain at certain grid planes, until each non-empty cell is a subbox of its own, leads to an appropriate decomposition. Different strategies have been developed to choose the splitting planes and will be discussed later. The decomposition is usually stored using a kd-tree [39, 134]. For each node at least the splitting plane and links to its two child nodes have to be stored. Each leaf corresponds to a subbox of the final decomposition.

Once the face through which a particle leaves a box is known, the next neighbor box can be accessed using the kd-tree. This requires an up traversal to the node which joins the branches of the corresponding leaves representing the current and the neighboring box. Then a down traversal is necessary, until the leaf node, which represents the neighboring subbox, is reached [10]. However, traversing upwards is only possible, if



(a) Spatial median splitting strategy. (b) Object median splitting strategy.

Figure 5.3: The computational effort for calculating a particle trajectory can be reduced by using a subdivision of the simulation domain into boxes. Empty grid cells are combined in larger boxes while non-empty grid cells (gray) are boxes by their own. There are various splitting strategies to obtain a suitable decomposition.

either the history of traversed nodes has been stored during down traversal, or, if links are available which give direct access to parent nodes.

The first approach is not very suitable for the previously presented rate calculation algorithm. Reemitted particles, which are launched from the surface intersection point, are buffered in a stack, while the trajectory of the incident particle is continued for a couple of grid spacings in order to consider potential disk intersections behind the surface. If only the initial position, which is the surface intersection, is stored with the new particles, the corresponding subbox must be re-determined before starting the trajectory calculation. To avoid the required costly complete top-down traversal, a reference to the subbox can be stored together with the position coordinates. Without parent links, the entire down traversal history would also have to be stored, which would slow down the entire simulation.

In the worst case a neighbor access within the kd-tree is of logarithmic complexity $\mathcal{O}(\log N_\mathrm{B})$. However, the neighbor box can be retrieved in constant time on average. Therefore the computational effort primarily scales with the average number of traversed subboxes.

5.2.7 Splitting Strategies

Different splitting strategies have been proposed to reduce the average number of traversed boxes [39, 72]. Among the simplest are the spatial median (SM) and the object median (OM) splitting strategy. The first algorithm chooses the splitting plane in such a way that the box is separated into two boxes of approximately equal size. The second algorithm uses the OM instead, which attempts to create boxes containing

approximately the same number of objects. The objects are in this case the non-empty cells. Figure 5.3 shows the corresponding spatial subdivisions for both strategies.

A much better strategy was presented in [72], which explicitly attempts to reduce the average number of traversed subboxes. The surface area heuristic (SAH) strategy uses a cost model to decide which splitting plane is best. For the following considerations all rays are assumed to be uniformly distributed. This corresponds to an arriving flux distribution at $\mathcal{P}$, which obeys the most frequently used directional distribution, the cosine distribution (2.5). Furthermore, the cost for a complete traversal through $\mathcal{B}$ is considered, which is not the case in reality, because particles are only tracked, until they reach the surface.

According to geometric probability theory [102], a uniformly distributed ray which intersects $\mathcal{B}$, also intersects another axis-aligned box $\mathcal{B}_i$ with $\mathcal{B}_i \subseteq \mathcal{B}$ with a probability of

$$p(\mathcal{B}_i \mid \mathcal{B}) = \frac{\mathrm{SA}(\mathcal{B}_i)}{\mathrm{SA}(\mathcal{B})}. \tag{5.22}$$

Here $\mathrm{SA}(\mathcal{B})$ and $\mathrm{SA}(\mathcal{B}_i)$ denote the surface areas of boxes $\mathcal{B}$ and $\mathcal{B}_i$, respectively. Hence the average number of traversed subboxes is given by

$$\sum_{i=1}^{N_\mathrm{B}} p(\mathcal{B}_i \mid \mathcal{B}) = \frac{\sum_{i=1}^{N_\mathrm{B}} \mathrm{SA}(\mathcal{B}_i)}{\mathrm{SA}(\mathcal{B})}. \tag{5.23}$$

The SAH strategy attempts to choose the splitting planes in such a way that this expression is minimized. Finding the absolute minimum of (5.23) is usually not possible in reasonable time. Instead, before subdividing a box, the additional costs are estimated in order to choose the best splitting plane.

Consider a box $\mathcal{B}_X$ during setup of the spatial subdivision, which needs to be split further. This is the case, if $\mathcal{B}_X$ is still larger than a grid cell and contains at least one non-empty cell. Let $N_X \geq 1$ be the number of non-empty grid cells in $\mathcal{B}_X$ which is subdivided along a certain plane into boxes $\mathcal{B}_Y$ and $\mathcal{B}_Z$, which then contain N_Y and N_Z (with $N_Y + N_Z = N_X$) non-empty grid cells, respectively. The additional costs are reduced, if the expression

$$N_Y \cdot \mathrm{SA}(\mathcal{B}_Y) + N_Z \cdot \mathrm{SA}(\mathcal{B}_Z) \tag{5.24}$$

is minimized [39, 72, 134]. Due to the restriction that boxes are only split along grid planes, it is possible to evaluate (5.24) for all potential splitting planes and to select the splitting plane with the minimum value. If the extent of box $\mathcal{B}_X$ in the x_i-direction (in terms of grid spacings) is denoted by L_i, the number of potential splitting planes is $\sum_{i=1}^{D} (L_i - 1)$. Subdivisions are favorable, which result in empty subboxes, because they do not need to be split further. For that purpose, expression (5.24) can be multiplied by a constant weight factor $\chi \leq 1$, if and only if $N_Y = 0 \vee N_Z = 0$ is true for the corresponding splitting plane [134].

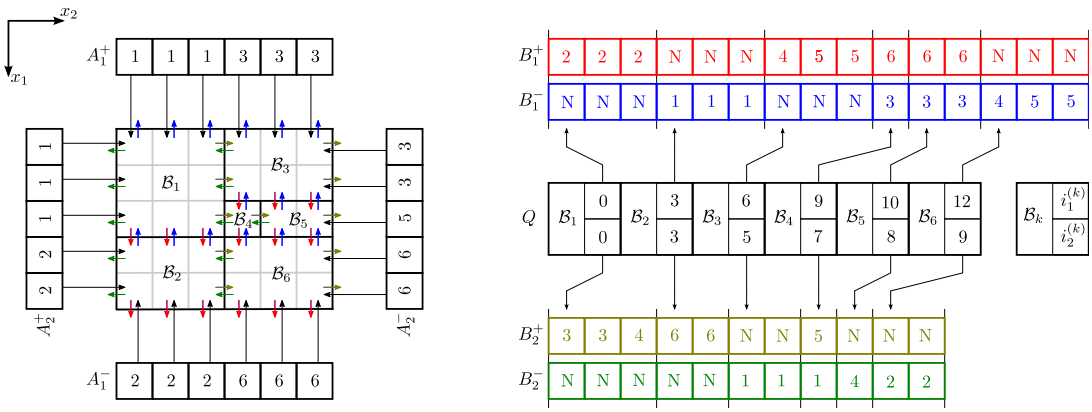


Figure 5.4: The arrays B_l^+ and B_l^- with $l \in \{1, 2\}$ store the neighbor links of all subboxes for the positive and negative x_l-direction, respectively. The array indices $i_1^{(k)}$ and $i_2^{(k)}$, which are stored together with box $\mathcal{B}_k$ in array Q, give access to the corresponding links. The additional arrays $A_l^{\pm}$ with $l \in \{1, 2\}$ allow fast access from the outside.

For good splitting strategies, such as the SAH, a logarithmic scaling with the number of objects can be observed for the average number of traversed boxes (5.23) [39]. Here the objects correspond to the non-empty cells which scale linearly with the surface size. Hence, as already stated in Section 5.2.4, the expected computational complexity of ray tracing is in the order of $\mathcal{O}(N_\mathrm{S} \log N_\mathrm{S})$ where N_S is a measure of the surface size.

5.2.8 Neighbor Links Arrays

To reduce the number of hierarchical traversals within a tree, direct neighbor links to subboxes have been proposed [72]. Apart from single neighbor links between nodes at the same level of the kd-tree, neighbor links trees can be used [39] to link leaves which correspond to subboxes, directly. A tree is set up for each face of a subbox to enable direct traversals to neighboring subboxes. Hence, using the exit point the next traversed subbox can be obtained by querying the corresponding neighbor links tree.

In the following discussion a new neighbor links data structure is presented. Using the fact that splitting planes are always aligned with the grid, arrays can be used to store the links to neighbor boxes, as demonstrated for a two-dimensional example in Figure 5.4. The number of links which are required for box $\mathcal{B}_k$, is equal to its surface area $\mathrm{SA}(\mathcal{B}_k)$ (measured in grid spacings). $2D$ different arrays $B_1^{\pm}, \ldots, B_D^{\pm}$ hold the neighbor links in a certain grid direction for all boxes. The number of links is equal for opposite directions. Hence, for each box $\mathcal{B}_k$ only D array indices $i_1^{(k)}, \ldots, i_D^{(k)}$ have to be stored in order to obtain the corresponding neighbor links. The size of the array

$B_l^{\pm}$ is given by

$$\dim(B_l^{\pm}) = \sum_{k=1}^{N_\mathrm{B}} \mathrm{FA}_l(\mathcal{B}_k) \tag{5.25}$$

where $\mathrm{FA}_l(\mathcal{B}_k)$ is the area of the face for which the normal points in the x_l-direction:

$$\mathrm{FA}_l(\mathcal{B}_k) := \prod_{\substack{i=1 \\ i\neq l}}^{D} \left(b_{\mathrm{max},i}^{(k)} - b_{\mathrm{min},i}^{(k)} \right). \tag{5.26}$$

Here $\vec{b}_{\mathrm{min}}^{(k)} \in \mathbb{Z}^D$ and $\vec{b}_{\mathrm{max}}^{(k)} \in \mathbb{Z}^D$ are the minimum and maximum index vectors of the corner grid points of box $\mathcal{B}_k$. The index number of the neighboring box at point $\vec{x} \in \mathbb{R}^D$, where the ray leaves $\mathcal{B}_k$ in the positive or negative x_l-direction, is given by

$$B_l^{\pm}\left[i_l^{(k)} + \Lambda_l^{(k)}(\lfloor x_{l+1} \rfloor, \ldots, \lfloor x_{l+D-1} \rfloor) \right]. \tag{5.27}$$

Here all subscripts are assumed to be cyclic modulo D plus 1 and $\lfloor \cdot \rfloor$ denotes the floor function. The bijective function

$$\Lambda_l^{(k)} : \prod_{i=l+1}^{l+D-1} \{ b_{\mathrm{min},i}^{(k)}, \ldots, b_{\mathrm{max},i}^{(k)} - 1 \} \leftrightarrow \{ 0, \ldots, \mathrm{FA}_l(\mathcal{B}_k) - 1 \} \tag{5.28}$$

is defined as

$$\Lambda_l^{(k)}(p_{l+1}, \ldots, p_{l+D-1}) := \sum_{i=l+1}^{l+D-1} \left(p_i - b_{\mathrm{min},i}^{(k)} \right) \prod_{j=l+1}^{i-1} \left(b_{\mathrm{max},j}^{(k)} - b_{\mathrm{min},j}^{(k)} \right). \tag{5.29}$$

The inverse function of $\Lambda_l^{(k)}$ is denoted as $\bar{\Lambda}_l^{(k)}$.

Additional links from outside of $\mathcal{B}$ allow finding the box which must be traversed first. They are stored in the $2D$ arrays $A_1^{\pm}, \ldots, A_D^{\pm}$ with size

$$\dim(A_l^{\pm}) = \mathrm{FA}_l(\mathcal{B}). \tag{5.30}$$

The first subbox which is traversed by a ray entering the box $\mathcal{B}$ at point $\vec{x}$ in the positive or negative x_l-direction, is given by

$$A_l^{\pm}\left[\Lambda_l(\lfloor x_{l+1} \rfloor, \ldots, \lfloor x_{l+D-1} \rfloor) \right]. \tag{5.31}$$

The memory requirements of the neighbor links array data structure are primarily given by the total surface area of all boxes $\sum_{k=1}^{N_\mathrm{B}} \mathrm{SA}(\mathcal{B}_k)$. An upper bound estimation can be derived for this sum using (5.23) and the fact that the typical ray traversal costs are $\mathcal{O}(\log N_\mathrm{S})$

$$\sum_{k=1}^{N_\mathrm{B}} \mathrm{SA}(\mathcal{B}_k) \leq \mathcal{O}(\log N_\mathrm{S}) \cdot \mathrm{SA}(\mathcal{B}). \tag{5.32}$$

For common problems the surface area of the bounding box $\mathcal{B}$ scales with the surface size. Therefore, a $\mathcal{O}(N_\mathrm{S} \log N_\mathrm{S})$ scaling law can be expected.

Once an appropriate spatial subdivision is found for $\mathcal{B}$, the neighbor links data structure can be set up. For subdivisions which are obtained by binary splitting in depth-first order, the neighbor links arrays data structure can be set up in optimal time (directly proportional to the total memory requirements) using Algorithm 5.1.

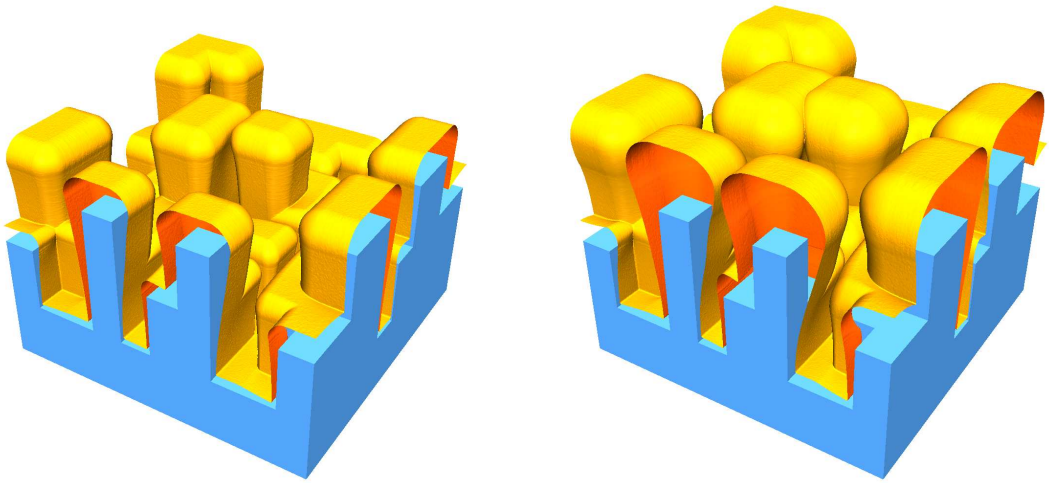
The neighbor links data structure allows an easy treatment of periodic boundary conditions. The boxes which are situated at the boundary can be directly linked to the corresponding boxes at the opposite boundary.

Algorithm 5.1 Setup of the neighbor links array data structure.

Require: $Q = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{N_\mathrm{B}}\}$ sorted in depth-first order
  for $l \leftarrow 1, D$ **do**
    determine sizes of arrays $A_l^\pm$ and $B_l^\pm$ given by (5.30) and (5.25), respectively
    allocate memory for arrays $A_l^\pm$ and $B_l^\pm$ and initialize to null
    $n \leftarrow 0$ ▷ index of current end within $B_l^\pm$
    for $k \leftarrow 1, N_\mathrm{B}$ **do** ▷ for all subboxes do
      for $m \leftarrow 0, \mathrm{FA}_l(\mathcal{B}_k) - 1$ **do**
        $i_l^{(k)} \leftarrow n$
        $m' \leftarrow \Lambda_l\left(\bar{\Lambda}_l^{(k)}(m)\right)$ ▷ map index from $\mathcal{B}_k$ to $\mathcal{B}$
        $B_l^-\left[i_l^{(k)} + m\right] \leftarrow A_l^-\left[m'\right]$
        if $A_l^+\left[m'\right] = \mathrm{null}$ **then**
          $A_l^+\left[m'\right] \leftarrow k$
        else
          $k' \leftarrow A_l^-\left[m'\right]$
          $m'' \leftarrow \Lambda_l^{(k')}\left(\bar{\Lambda}_l^{(k)}(m)\right)$ ▷ map index from $\mathcal{B}_k$ to $\mathcal{B}_{k'}$
          $B_l^+\left[i_l^{(k')} + m''\right] \leftarrow k$
        end if
        $A_l^-\left[m'\right] \leftarrow k$
      end for
      $n \leftarrow n + \mathrm{FA}_l(\mathcal{B}_k)$
    end for
  end for

5.2.9 Parallelization

The parallelization of the rate calculation using ray tracing is very simple on shared memory architectures. Due to the assumption of ballistic transport, particle trajectories are independent of each other. The total number of particles can simply be



(a) After 50 time units. (b) After 100 time units.

Figure 5.5: A deposition process simulation was used for the benchmarks.

distributed over multiple CPUs. Basically, this can be realized through parallelization of a loop, which is straightforward when using OpenMP [23]. To avoid simultaneous write accesses to the surface rates, individual arrays are used for each CPU, which are finally summed up.

5.2.10 Benchmarks

To demonstrate the ray tracing technique for surface rate calculation, a deposition process was applied to the initial structure given in Figure 4.5a. The structure was resolved on a grid with lateral grid extensions 500×500. A total of 25 million particles were simulated at every time step. The arrival flux distribution at the source plane was assumed to follow (2.5) with $F_{\text{neu}}^{\text{src}} = 1$. The sticking probability was 0.5 and diffusive reemission was assumed (2.18). For each incident particle, a new one was reemitted as long as the weight factor of the new particle was larger than 0.01 ($w > 0.01$, see Section 5.2).

For the CFL criterion required by the LS method, $C_{\text{CFL}} = 0.1$ was chosen. The results after a process time of 50 and 100 time units are shown in Figure 5.5. The simulation was carried out on 16 cores of AMD Opteron 8435 processors (2.6 GHz). Different ray tracing data structures and splitting strategies were tested. Table 5.1 summarizes the average times which are spent on setting up the data structure and on ray tracing at each time step. The average memory consumption is also listed.

Best runtimes have been observed for the SAH with $\chi = 0.8$. It should be noted that the measured times also include the costs for disk and surface intersection tests and

Data structure	Splitting strategy	Setup time	Calc. time	Memory costs
Full grid	–	0.18 s	146.0 s	235.5 MB
Kd-tree	SM	0.77 s	74.3 s	172.9 MB
	OM	1.07 s	70.5 s	135.9 MB
	SAH ($\chi = 0.6$)	2.06 s	59.1 s	129.4 MB
	SAH ($\chi = 0.7$)	1.98 s	58.6 s	129.9 MB
	SAH ($\chi = 0.8$)	1.98 s	60.2 s	131.5 MB
	SAH ($\chi = 0.9$)	2.00 s	60.4 s	134.8 MB
	SAH ($\chi = 1.0$)	2.09 s	63.8 s	148.4 MB
Neighbor links arrays	SM	1.79 s	67.2 s	258.9 MB
	OM	1.94 s	59.8 s	214.1 MB
	SAH ($\chi = 0.6$)	2.80 s	53.6 s	188.1 MB
	SAH ($\chi = 0.7$)	2.69 s	52.5 s	181.9 MB
	SAH ($\chi = 0.8$)	2.65 s	52.3 s	181.2 MB
	SAH ($\chi = 0.9$)	2.68 s	53.6 s	184.1 MB
	SAH ($\chi = 1.0$)	2.83 s	55.3 s	197.0 MB

Table 5.1: Comparison of different data structures for ray tracing. All tests were carried out on 16 cores of AMD Opteron 8435 processors (2.6 GHz).

for the random direction generation, which are identical for all data structures. Hence, the computational savings due to the ray tracing data structures are manifested by the measured values.

The results clearly show that the ray tracing data structures are superior to the full grid which exhibits a much longer calculation time. The full grid stores a single link for every cell. In the case of a non-empty cell, this link gives access to its data. The bad scaling behavior, which is linear with the domain size, leads to a high memory consumption for larger structures.

The memory requirements for trees are much lower, because they are expected to scale linearly with the surface size. The applied ray tracing algorithm requires for each leaf the minimum and maximum indices of its bounding box. Furthermore, links to the two child nodes as well as the position and orientation of the splitting plane are needed. Finally, parent links must be stored for all nodes to enable up traversals within the tree.

The neighbor links arrays cannot compete with the kd-tree concerning memory consumption. However, the former shows better runtimes for most tested geometries. A likely reason for the good runtime performance is that the data structure clearly separates the neighbor links for positive and negative grid directions. Hence, depending on the direction of the ray, only the corresponding positive or negative array must be accessed. Since runtime, and not memory consumption, is the major concern for most topography simulations using ray tracing, the neighbor links arrays using SAH with $\chi = 0.8$ are used for all simulations presented in Chapter 6.

	AMD Opteron 8435 (6 × 2.6 GHz)		AMD Opteron 8222 SE (2 × 3 GHz)	
Num. CPUs	Calc. Time	Efficiency	Calc. Time	Efficiency
1	505.2 s	100.0 %	624.8 s	100.0 %
2	314.8 s	80.2 %	332.6 s	93.9 %
4	177.0 s	71.4 %	182.1 s	86.0 %
8	93.0 s	67.9 %	101.3 s	77.1 %
16	52.3 s	60.4 %	–	–

Table 5.2: Parallel scalability of the example shown in Figure 5.5. The neighbor links arrays data structure using the SAH with $\chi = 0.8$ was used for these benchmarks.

The parallel efficiency of the surface rate calculation procedure was also tested on machines with four AMD Opteron 8435 six-core processors and AMD Opteron 8222 SE dual-core processors clocking at 2.6 GHz and 3 GHz, respectively. The average calculation time for rate calculation, using different numbers of cores, are listed in Table 5.2. The numbers refer to the neighbor links arrays data structure using SAH with $\chi = 0.8$.

The parallel ray tracing algorithm is essentially just a loop whose independent iterations are distributed over all CPUs. Therefore, losses due to synchronization of threads can be neglected. Furthermore, memory access is read-only except for the surface rates. Each core has its own copy of the surface rates, which are finally summed up, in order to avoid concurrent write accesses. As a consequence, the memory bandwidth is the main crucial factor for the parallel efficiency, due to the random memory access pattern of the ray tracing algorithm. The efficiency is worse on the machine using six-core processors than on that using dual-core processors, because more cores must share the common connection to the main memory. This bottleneck is a well-known problem of today's multi-core processors [130].

5.3 Generation of Random Vectors

The generation of random directions is essential for the Monte Carlo based calculation of particle transport. It is difficult to find instructions for the generation of random directions which obey a certain angular distribution, except for uniform spherical [92] or cosine distributions [37]. Therefore, recipes for the generation of variate distributions used in this work are given in the following discussion.

The arrival directions of particles at the source plane $\mathcal{P}$ and the directions of reemitted particles are usually described by probability densities $f(\vec{\omega})$, which only depend on the polar angle θ relative to a certain direction $\vec{v}$

$$f(\vec{\omega}) = g(\theta) \quad \text{with } \theta = \arccos(\vec{v} \cdot \vec{\omega}). \tag{5.33}$$

Using spherical coordinates with respect to $\vec{v}$ the probability density function can be formulated as

$$f(\vec{\omega})\, d\Omega = g(\theta) \sin\theta \, d\theta \, d\phi. \tag{5.34}$$

Since this expression is separable, the azimuthal angle ϕ and the polar angle θ are independent, which allows for the description of both variables by individual probability densities $f_\phi(\phi)$ and $f_\theta(\theta)$

$$f(\vec{\omega})\, d\Omega = f_\phi(\phi)\, d\phi \cdot f_\theta(\theta)\, d\theta \quad \text{with } f_\phi(\phi) = 1 \text{ and } f_\theta(\theta) = g(\theta)\sin\theta. \tag{5.35}$$

The probability density function f_ϕ is constant, which is a direct consequence of the rotational symmetry of the directional distribution. Therefore, since the azimuthal angle is uniformly distributed on $[0, 2\pi[$, a random choice of ϕ is trivial. Picking a random polar angle is more sophisticated. In the following sections algorithms are presented for selecting a polar angle according to directional distributions which are frequently used for the description of arrival or reemission angles.

5.3.1 Power Cosine Distribution

First the power cosine distribution as introduced in (2.6) is considered, where $g(\theta) = (\cos\theta)^\nu$ with $\nu > 0$. Thus the probability density function for the polar angle is given as

$$f_\theta(\theta) = (\cos\theta)^\nu \sin\theta. \tag{5.36}$$

Calculating the cumulative distribution function results in

$$F_\theta(\theta) = \frac{\int_0^\theta f_\theta(\theta')\, d\theta}{\int_0^{\frac{\pi}{2}} f_\theta(\theta')\, d\theta} = 1 - (\cos\theta)^{\nu+1}\,. \tag{5.37}$$

A variate obeying an arbitrary distribution can be obtained using the inversion method [25]. A uniformly distributed variate u on $[0, 1]$ mapped by the inverse cumulative distribution function leads to the desired distribution [115]

$$\theta = F_\theta^{-1}(u) = \arccos\left(\sqrt[\nu+1]{1-u}\right). \tag{5.38}$$

Since $1 - u$ is uniformly distributed on $[0, 1]$ as well, the random polar angle can also be generated by [66]

$$\theta = \arccos\left(\sqrt[\nu+1]{u}\right). \tag{5.39}$$

Algorithm 5.2 summarizes the generation of a random polar angle following the probability density function (5.36).

Algorithm 5.2 Generation of a $(\cos\theta)^\nu \sin\theta$ distributed variate.

function PowerCosineVariate(ν) ▷ requires $\nu > -1$
  return $\arccos\left(\sqrt[\nu+1]{\text{rand}()}\right)$ ▷ rand() returns uniform random number on $[0, 1]$
end function ▷ Note: If just $\cos\theta$ is required return $\sqrt[\nu+1]{\text{rand}()}$ instead.

5.3.2 Coned Cosine Distribution

Next, the distribution introduced in (2.17) is considered, for which $g(\theta) = \cos(\theta/a)$ for all $\theta \in [0, \theta_{\text{cone}}]$. Hence, all angles are within a cone with apex angle $2\theta_{\text{cone}}$. This is the reason for the name of the distribution used in this work. a and θ_{cone} are related by $\frac{\pi}{2}a = \theta_{\text{cone}}$. To restrict the emission of particles to one hemisphere, θ_{cone} must satisfy $\theta_{\text{cone}} \leq \frac{\pi}{2}$, which is equivalent to $a \leq 1$. The probability density of the polar angle is given by

$$f_\theta(\theta) = \cos(\theta/a)\sin\theta. \tag{5.40}$$

The corresponding cumulative distribution function is

$$F_\theta(\theta) = \begin{cases} \frac{\sin\theta\sin(\theta/a)+a(\cos\theta\cos(\theta/a)-1)}{\sin(\frac{\pi}{2}a)-a} & a < 1, \\ (\sin\theta)^2 & a = 1, \end{cases} \tag{5.41}$$

Since it is not possible to calculate an explicit expression for the inverse function $F_\theta^{-1}(\theta)$, which is useful for the inverse method, the rejection technique is chosen instead [25]. For the rejection method an instrumental distribution $f'_\theta(\theta)$ is necessary, which is an upper bound approximation of $f_\theta(\theta)$, and which leads to an invertable cumulative distribution function.

Using the inequalities

$$\cos x \leq 1 - \left(\tfrac{2}{\pi}x\right)^2 \quad \forall x \in \left[-\tfrac{\pi}{2}, \tfrac{\pi}{2}\right] \tag{5.42}$$

and

$$\sin x \leq x \quad \forall x \geq 0 \tag{5.43}$$

(see Inequality 1 and Inequality 2 in Appendix C) such an instrumental probability density function for (5.40) is given by

$$f'_\theta(\theta) := \left(1 - \frac{\theta^2}{\theta_{\text{cone}}^2}\right) \cdot \theta \geq f_\theta(\theta) \quad \forall\theta \in [0, \theta_{\text{cone}}]\,. \tag{5.44}$$

The corresponding cumulative distribution function is

$$F'_\theta(\theta) = 1 - \left(1 - \left(\frac{\theta}{\theta_{\text{cone}}}\right)^2\right)^2. \tag{5.45}$$

According to the rejection method a random polar angle following f_θ can be generated by feeding the inverse of the instrumental cumulative distribution function

$$\theta = F'^{-1}_\theta(u) = \theta_{\text{cone}}\sqrt{1-\sqrt{1-u}} \tag{5.46}$$

with uniformly distributed random variates $u \in [0,1]$ and accepting θ, if

$$u' f'_\theta(\theta) \leq f_\theta(\theta), \tag{5.47}$$

where u' denotes another uniformly distributed variate on $[0,1]$. Algorithm 5.3 summarizes the entire procedure for determining random polar angles, which are distributed according to the coned cosine distribution.

Algorithm 5.3 Generation of a $\cos(\frac{\pi}{2}\theta/\theta_{\text{cone}})\sin\theta$ distributed variate.

```
function ConedCosineVariate(θ_cone)                    ▷ requires 0 ≤ θ_cone ≤ π/2
    repeat
        x ← √rand()
        y ← √(1 − x)
        θ ← θ_cone · y
    until x · θ · rand() ≤ cos(π/2 · y) · sin θ
    return θ                                           ▷ θ satisfies 0 ≤ θ ≤ θ_cone
end function
```

The efficiency γ of rejection sampling for this case, thus the fraction of successful attempts satisfying (5.47), is given by

$$\gamma = \frac{\int_0^{\theta_{\text{cone}}} f_\theta(\theta')\,d\theta}{\int_0^{\theta_{\text{cone}}} f'_\theta(\theta')\,d\theta} = \frac{8}{\pi^2}\cdot\begin{cases}\frac{2\left(\sin\left(\frac{\pi}{2}a\right)-a\right)}{a(1-a^2)} & \text{if } 0<a<1,\\ 1 & \text{if } a=1.\end{cases} \tag{5.48}$$

The presented algorithm shows a very high efficiency, since $\gamma \geq \frac{8}{\pi^2}$ holds for all $a \in \,]0,1]$ (see Inequality 3 in Appendix C), which corresponds to a success rate of approximately 81 % in the worst case.

5.3.3 Direction Vector Calculation

Ray tracing usually requires a normalized direction vector. Therefore, once random polar and azimuthal angles with respect to the given unit vector $\vec{v}$ are determined, the direction vector $\vec{\omega}$ can be obtained by

$$\vec{\omega} = \vec{v}\cos\theta + \left(\vec{v}'\cos\phi + \vec{v}\times\vec{v}'\sin\phi\right)\sin\theta, \tag{5.49}$$

where $\vec{v}'$ is a normal vector with respect to $\vec{v}$, thus satisfying $\vec{v}'\cdot\vec{v}=0$. The rotational symmetry of the random direction distribution allows an arbitrary choice of $\vec{v}'$, which

can be defined as follows

$$\vec{v}' := \begin{cases} \frac{1}{\sqrt{1-v_1^2}} \cdot (0, v_3, -v_2) & |v_1| \leq |v_2|, \\ \frac{1}{\sqrt{1-v_2^2}} \cdot (v_3, 0, -v_1) & |v_1| > |v_2|. \end{cases} \tag{5.50}$$

The case differentiation avoids problematic cases, where $\vec{v}'$ vanishes. In the following, only the first case is considered. The second case can be converted to the first case by exchanging the first two components of $\vec{v}$, v_1 and v_2, before calculating $\vec{\omega}$. A final exchange of the corresponding components of $\vec{\omega}$, ω_1 and ω_2, leads to the correct direction vector.

The condition $|v_1| \leq |v_2|$ together with $\|\vec{v}\| = 1$ implies $|v_1| \leq \frac{1}{\sqrt{2}}$ and $|v_2| \geq \frac{1}{\sqrt{3}} \vee |v_3| \geq \frac{1}{\sqrt{3}}$. Therefore, $\vec{v}'$ is always well-defined. Insertion into (5.49) gives

$$\begin{aligned} \omega_1 &= v_1 \cos\theta - f\left(1 - v_1^2\right) \sin\phi \\ \omega_2 &= v_2 \left(\cos\theta + f\, v_1 \sin\phi\right) + f\, v_3 \cos\phi \\ \omega_3 &= v_3 \left(\cos\theta + f\, v_1 \sin\phi\right) - f\, v_2 \cos\phi \end{aligned} \tag{5.51}$$

with $f := \frac{\sin\theta}{\sqrt{1-v_1^2}} = \sqrt{\frac{1-(\cos\theta)^2}{1-v_1^2}}$. Using the second notation of f, $\vec{\omega}$ can be calculated without knowledge of $\sin\theta$ at the expense of one additional multiplication. Some distributions, such as the previously discussed power cosine distribution, enable the direct calculation of $\cos\theta$, avoiding the costly evaluation of trigonometric functions for the polar angle entirely (see the last note in Algorithm 5.2).

The evaluation of the other trigonometric functions in (5.51) can also be avoided. The point $(\sin\phi, \cos\phi)$ is uniformly distributed on the unit circle. An alternative way for picking a point on the unit circle is to randomly choose a point (a_1, a_2) on a disk and to calculate the normalized vector $(a_1, a_2)/\sqrt{a_1^2 + a_2^2}$ [25]. The radicand can be combined with that for the calculation of f obviating the extra evaluation of the root. Algorithm 5.4 describes the determination of a random unit vector $\vec{\omega}$ around $\vec{v}$ with given polar angle θ, which is equivalent to $\vec{\omega} \cdot \vec{v} = \cos\theta$.

5.3.4 Cosine Distribution

The most frequently used distribution is the cosine distribution, which is required to describe the arrival angles of neutral particles (2.5) or diffusive reemission (2.18). The cosine distribution can be seen as a special case of the power cosine distribution (2.6) with $\nu = 1$ or as a special case of the coned cosine distribution with $\theta_{\text{cone}} = \frac{\pi}{2}$. As shown in the next section, sampling a cosine distribution using Algorithm 5.2 is faster than using Algorithm 5.3 for the coned cosine distribution. Therefore, the power cosine distribution with $\nu = 1$ is the common technique for sampling a cosine distribution [37]. However, the polar angle is usually not of interest. It is just an intermediate result passed to Algorithm 5.4 to generate a unit vector as needed for ray tracing.

Algorithm 5.4 Sampling a random vector $\vec{\omega}$ satisfying $\vec{\omega} \cdot \vec{v} = \cos\theta$.

function RandomUnitVector($\vec{v}, \cos\theta$) ▷ requires $\|\vec{v}\|$=1 and $|\cos\theta| \leq 1$
  repeat ▷ pick uniform random point inside circle with radius 0.5
    $a_1 \leftarrow \text{rand}() - 0.5$
    $a_2 \leftarrow \text{rand}() - 0.5$
    $a \leftarrow a_1^2 + a_2^2$
  until $0 < a \leq 0.25$
  if $|v_1| \leq |v_2|$ **then**
    $v_1 \leftrightarrow v_2$ ▷ swap v_1 and v_2
    set flag
  end if
  $b \leftarrow 1 - v_1^2$
  $c \leftarrow \sqrt{\frac{1-(\cos\theta)^2}{a \cdot b}}$
  $a_1 \leftarrow c \cdot a_1$
  $a_2 \leftarrow c \cdot a_2$
  $d \leftarrow \cos\theta - v_1 \cdot a_1$
  $\omega_1 \leftarrow v_1 \cdot \cos\theta - b \cdot a_1$
  $\omega_2 \leftarrow v_2 \cdot d + v_3 \cdot a_2$
  $\omega_3 \leftarrow v_3 \cdot d - v_2 \cdot a_2$
  if flag is set **then**
    $\omega_1 \leftrightarrow \omega_2$ ▷ swap ω_1 and ω_2
  end if
  return $\vec{\omega}$
end function

Algorithm 5.5 Picking a random point on unit sphere.

function RandomPointOnUnitSphere()
  repeat
    $a_1 \leftarrow 2 \cdot \text{rand}() - 1$
    $a_2 \leftarrow 2 \cdot \text{rand}() - 1$
    $a \leftarrow a_1^2 + a_2^2$
  until $a < 1$
  $\omega_1 \leftarrow 1 - 2 \cdot a$
  $a \leftarrow 2 \cdot \sqrt{1 - a}$
  $\omega_2 \leftarrow a_1 \cdot a$
  $\omega_3 \leftarrow a_2 \cdot a$
  return $\vec{\omega}$ ▷ $\vec{\omega}$ satisfies $\|\vec{\omega}\| = 1$
end function

Algorithm 5.6 Generation of a $\cos\theta$ distributed unit vector around $\vec{v}$.

function CosineDistributedRandomVector($\vec{v}$) ▷ requires $\|\vec{v}\| = 1$
  repeat
    $\vec{\omega}' \leftarrow$RandomPointOnUnitSphere()
    $\vec{\omega} \leftarrow \vec{\omega}' + \vec{v}$
    $a \leftarrow \sqrt{\omega_1^2 + \omega_2^2 + \omega_3^2}$
  until $a \neq 0$
  $\vec{\omega} \leftarrow \vec{\omega}/a$
  return $\vec{\omega}$
end function

In the following discussion an alternative method is proposed, which samples a unit vector obeying the cosine distribution directly, without the need of an additional rotation as described in Section 5.3.3. The method uses the fact that the sum of the given direction vector $\vec{v}$ and a unit random vector $\vec{\omega}'$, which is uniformly distributed over a sphere, follows a cosine distribution. As illustrated in Figure 5.6, the area dA seen from the origin O in a direction with polar angle θ within an infinitesimal solid angle $d\Omega$ is

$$dA = 4\cos\theta\, d\Omega. \tag{5.52}$$

As a consequence, the desired random vector $\vec{\omega}$ can be obtained from a spherically distributed unit vector $\vec{\omega}'$ by

$$\vec{\omega} = \frac{\vec{v} + \vec{\omega}'}{\|\vec{v} + \vec{\omega}'\|}. \tag{5.53}$$

For the spherical distribution numerous methods have been proposed in the past. So far the most efficient way to pick a point on the unit sphere is described in [61, 75, 107] and is listed in Algorithm 5.5. Standard numeric libraries such as the GNU scientific library already provide ready implementations for that algorithm [36]. Therefore, a very simple and short vector sampling algorithm for the cosine distribution is given by Algorithm 5.6.

5.3.5 Direction Vector Sampling Benchmarks

All the algorithms, presented in the previous sections, were implemented using C++ and tested on an Intel Core 2 Duo E6600 processor clocking at 2.4 GHz. To compare the algorithms for different parameters, $\theta_{1/2}$ was varied, which is the angle at half maximum $g(\theta_{1/2}) = \frac{1}{2}$. For the power cosine distribution $\theta_{1/2} = \arccos\left(\frac{1}{\sqrt[n]{2}}\right)$ and for the coned cosine distribution $\theta_{1/2} = \frac{2}{3}\,\theta_{\text{cone}}$.

The runtimes for sampling 100 million random vectors are compared for different algorithms in Table 5.3. For $\theta_{1/2}$ the random direction vectors follow a simple cosine distribution for all presented algorithms. For this specific case Algorithm 5.6 is able

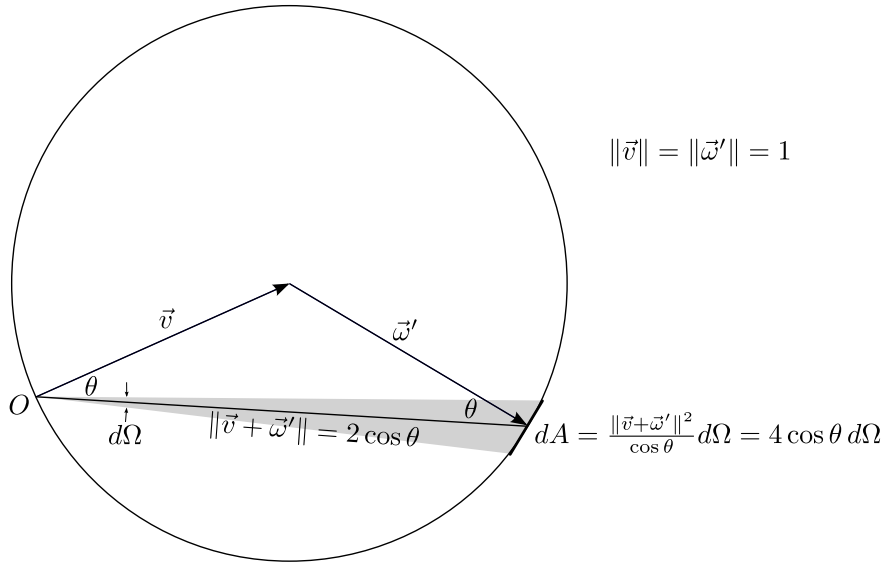


Figure 5.6: The sum of the given direction $\vec{v}$ and a random unit vector $\vec{\omega}'$ uniformly distributed over a sphere leads to a cosine distribution.

Angle at half max.	Power cosine		Coned cosine		Simple cosine
$\theta_{1/2}$	ν	t	θ_{cone}	t	t
60°	1	10.8 s	90°	26.6 s	11.6 s
21.1°	10	20.8 s	31.6°	26.8 s	–
6.74°	100	20.8 s	25.8°	25.8 s	–
2.13°	1000	20.8 s	3.20°	25.8 s	–

Table 5.3: Runtimes for sampling 100 million direction vectors on an Intel Core 2 Duo E6600 processor running at 2.4 GHz.

to compete with Algorithm 5.2 for the power cosine distribution with $\nu = 1$ and is therefore, due to its simplicity, a serious alternative. For $\nu = 1$ the compiler is able to use the square root function instead of the power function, which explains the better runtime. For all other cases, the runtimes for sampling the power cosine distribution are comparable with those for sampling the coned cosine distribution.

5.4 Implementation Details

The complete topography simulation algorithm was coded in C++. Most parts were implemented in a multi-dimensional way. Only a few functions were treated separately for the two- and three-dimensional cases, such as testing for ray–surface intersections, which requires bi- or trilinear interpolation in two and three dimensions, respectively.

However, using the template mechanism of C++ [124], both cases are processed separately during compile time. In this way, heavy optimizations by the compiler, such as loop unrolling for iterations over the dimensions, are possible.

5.4.1 Simulation Flow

The basic flow of the complete algorithm for topography simulation is depicted in Figure 5.7. First, the initial geometry is given as a volume mesh, where a material number is assigned to each volume element, which is either a triangle or a tetrahedron for two or three dimensions, respectively. The order of these numbers defines, in which way the structure is represented by LSs (see Section 4.6.1). The surface and the corresponding interfaces are then extracted, which results, depending on the number of dimensions, in triangle or line segmentations which are further transformed into LS representations.

After initialization, the main cycle of the algorithm is started. For all active grid points the tangential disks are set up. For each active grid point the distance to the disk and the normal vector are calculated and stored. Afterwards, the set of all non-empty grid cells is determined, which is then used to set up an appropriate spatial subdivision. The subsequent ray tracing procedure calculates the surface rates which are needed to calculate the surface velocities.

As previously discussed in Section 2.2.3, the reemission probability of particles might depend on the arriving flux distribution, which is the case for the non-linear surface reactions presented in Section 2.3.2. This leads to a recursive problem which can be solved through iterations. Hence, the computation of the surface rates must be repeated several times, until convergence is achieved. In practice, this is usually performed only for the first time step. Later, the reemission probability is estimated from the surface rates, which have been obtained in the preceding time step. This is a reasonable approach, since subsequent geometric changes are very small due to the CFL condition.

After the calculation of the surface rates, the surface velocity can be computed for all active grid points. Then, the surface is advanced using the LS algorithm and the time is incremented. If the elapsed time is still smaller than the total processing time, the entire time evolution cycle is repeated. Otherwise, the marching squares or marching cubes algorithm, depending on the number of dimensions, is applied in order to obtain an explicit surface representation as final output.

5.4.2 Model Implementation

The simulator is designed in such a way that the simulator kernel is clearly separated from the physical models. Hence, new models can be easily implemented and added.

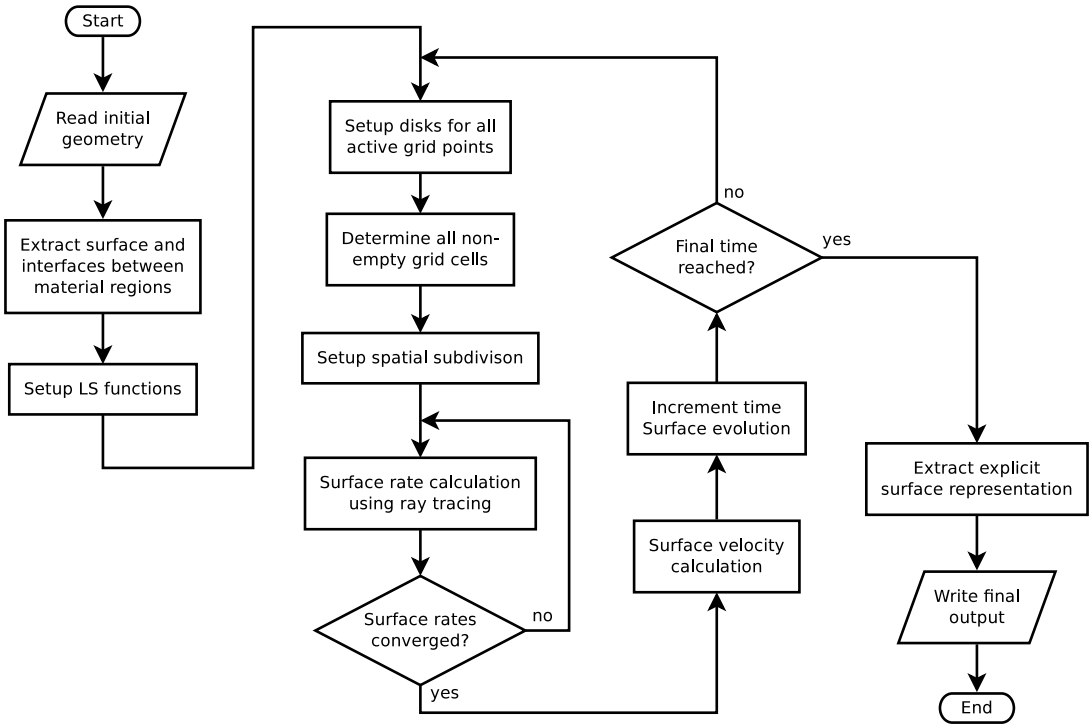


Figure 5.7: Flow chart of the simulation algorithm.

The simulator kernel requests 4 functions of specific form which all must be provided by a new process model:

1. The first function

   ```
   void particle_generation( double time,
                             double position[],
                             particle_t& particle );
   ```

   generates a new particle which obeys the arriving flux distribution Γ_{src}. It returns a starting position and the particle properties, such as direction, energy, and species. The particle properties are stored using the data type

   ```
   struct particle_t {
       double direction[3];
       double energy;
       int species;
       double weight_factor;
   };
   ```

   which is provided by the simulator and also includes a weight factor. The initial value of the weight factor is usually 1. The starting position does not need to be set by the function, if the arriving flux distribution is spatial invariant. In this

case a flag can be passed to the simulator kernel which itself chooses a random starting position on the source plane $\mathcal{P}$.

2. The second function

```
void disk_intersection( const double normal[],
                        double rates[],
                        const double old_rates[],
                        const particle_t& particle );
```

is called, whenever a particle intersects the tangential disk of an active grid point. If this is the case, the function is responsible to make the corresponding contributions to the surface rates. Apart from the normal vector of the tangential disk, the old rates from the previous iteration step are provided, which are needed to solve recursive problems.

3. The third function

```
void surface_intersection( int material,
                           const double normal[],
                           const double old_rates[],
                           const particle_t& particle,
                           stack<particle_t>& new_particles );
```

describes what needs to be done if a particle intersects the surface. The current material on the surface and the surface normal (obtained by deriving the multi-linear interpolation formula within the corresponding grid cell) at the intersection point are passed to the function. Furthermore, the preceding surface rates, as needed for recursive problems, and the material type are provided. Since the rates are only stored at active grid points, the rates of the closest active grid point are taken. Together with the properties of the incident particle all information necessary in deciding whether new particles must be launched, is available. The properties of new particles, such as the reemitted direction, the energy, the particle species, and the weight factor, which all must obey the reemission probability, are simply pushed to the stack. The starting position of new particles is automatically assumed to be the surface intersection point.

4. The fourth function

```
double velocity_calculation( int material,
                             const double normal[],
                             const double rates[]    );
```

is called, if the surface velocity must be known for an active grid point and for a certain material type. The normal vector is also passed to the function for the description of transport-independent surface reactions (see Section 2.3.3).

A new process model is represented as a class which contains all these functions in the public section. To simulate a certain process, the corresponding class is simply passed

to the simulation algorithm. C++ offers dynamic and static polymorphism. However, especially the first three functions are called very often during the ray tracing procedure. At the same time, they usually contain only a few lines of code which is predestined for inlining. Therefore, to enable compiler optimizations, static polymorphism is applied, which is expressed in C++ by passing the class of a physical model as a template parameter to the simulator kernel.

6 Applications

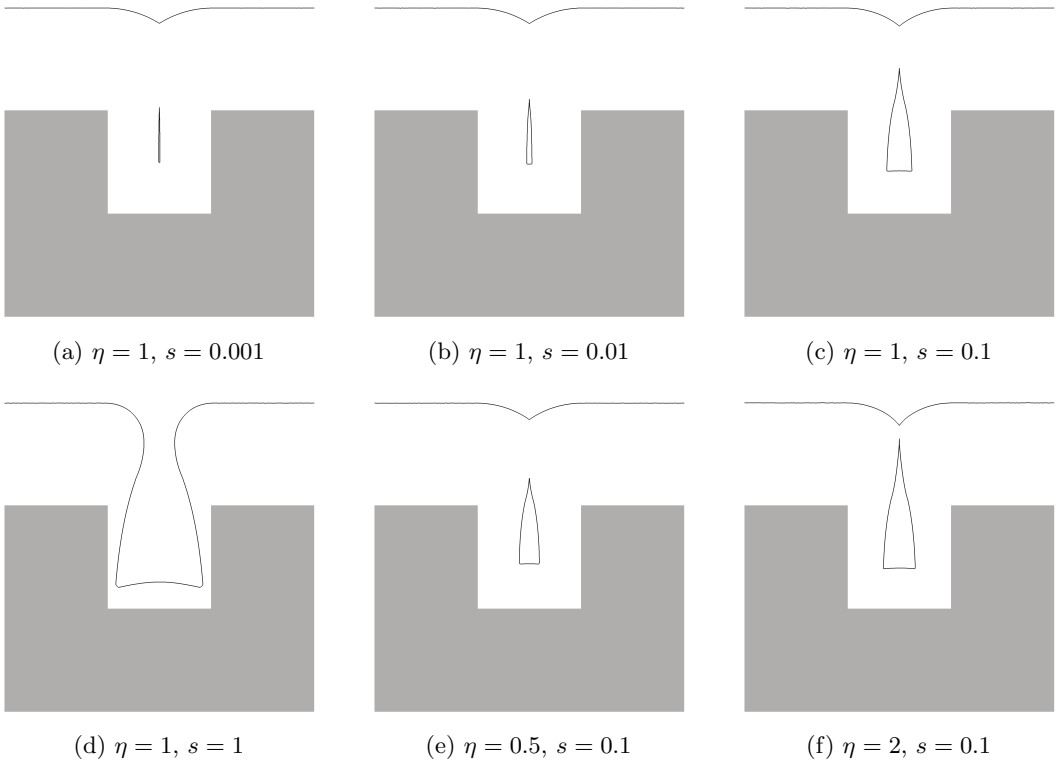
In this chapter all previously described numerical techniques are demonstrated on a selection of different processes which are covered by the general model equations introduced in Chapter 2. Due to the simple simulator interface (see Section 5.4.2) and the simulator's ability to solve the general transport equations without further simplifications, new models can be straightforwardly implemented. The physical models of processes, which were used for the simulations presented in the following, were all taken from literature.

6.1 Chemical Vapor Deposition

In [21], a simple model, which is able to describe low pressure CVD and PVD processes, was presented. It assumes ballistic particle transport at feature-scale and considers only a single particle species. Particles remain sticking on the surface according to a given sticking probability. Otherwise, the particles are reemitted from the surface, and their directions follow the Knudsen cosine law (2.18). For non-linear surface reactions, a dependence of the local sticking probability on the incident flux was derived (2.32), leading to a recursive problem.

First, in order to compare the results and to verify the simulator, some two-dimensional low pressure CVD simulations, initially published in [21] are reproduced. The initial structure is a quadratic trench. The directional distribution of arriving particles is also assumed to obey a cosine distribution (2.5). The results for various sticking coefficients s and reaction orders η are shown in Figure 6.1. The value of s corresponds, in the case of non-linear surface reaction ($\eta \neq 1$), to the sticking probability on a plane surface. The results are in good agreement with those given in [21].

The well behaving scaling laws of all the numerical techniques allow the simulation of large three-dimensional structures. In order to demonstrate the capabilities of the simulator, a deposition process is simulated for a large initial structure as shown in Figure 6.2. Its lateral extensions are 1400×835 grid spacings. The final profiles after deposition of a 15 and a 30 grid spacings thick layer are shown in Figure 6.3 and Figure 6.4, respectively. The deposition process was modeled using the parameters $\eta = 1$ and $s = 0.05$.



(a) $\eta = 1$, $s = 0.001$ (b) $\eta = 1$, $s = 0.01$ (c) $\eta = 1$, $s = 0.1$

(d) $\eta = 1$, $s = 1$ (e) $\eta = 0.5$, $s = 0.1$ (f) $\eta = 2$, $s = 0.1$

Figure 6.1: Two-dimensional simulations of a deposition process for various sticking coefficients s and reaction orders η.

For the simulation 1168 million particles are simulated at every time step in order to calculate the local deposition rates. Every time a particle hits the surface, a new particle is launched, as long as its new weight factor is larger than 0.001. The simulation was carried out in parallel on 24 cores of AMD Opteron 8435 processors (2.4 GHz). The average calculation time for a time step is 10 min. In total, 600 time steps are necessary for the entire simulation, when using $C_{\mathrm{CFL}} = 0.1$ for the CFL criterion.

6.2 Plasma Etching

In recent years various works on three-dimensional plasma etching simulation have been presented. However, most of them are not suitable to solve complex plasma etching models. The transport equations of more generalized models cannot be solved in three dimensions by the conventional approach, due to computational limitations. For simplification, specular reflections of ions or higher order reemissions of neutrals are often neglected [43, 50]. In order to incorporate such effects, particle trajectories must be calculated using the MC technique [64, 95]. The ray tracing techniques described

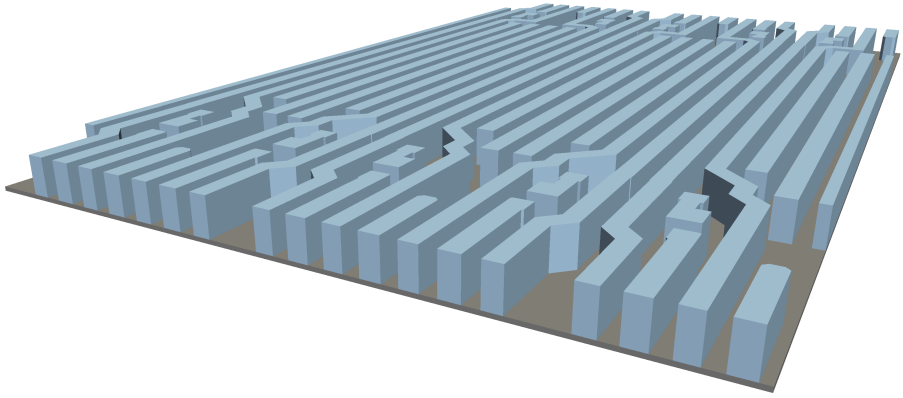


Figure 6.2: The initial geometry was resolved on a grid with lateral extensions 1400×835.

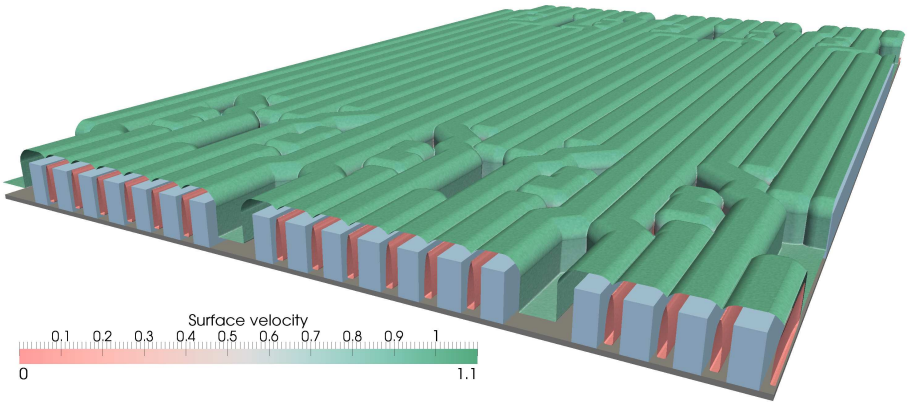


Figure 6.3: The profile after deposition of a 15 grid spacings thick layer.

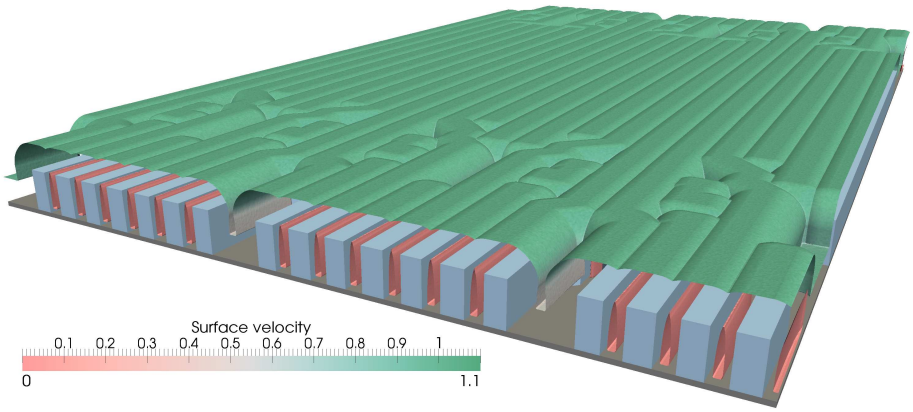


Figure 6.4: The profile after deposition of a 30 grid spacings thick layer.

in this work derive an efficient solution, even for large three-dimensional structures [A19].

Typical plasma etching models assume ballistic transport of particles at feature-scale and Langmuir-type adsorption [1, 14, 15, 63]. As a representative of these mathematically similar models, the SF_6/O_2 plasma etching model given in [14] is selected. The surface kinetics model and its governing equations are briefly discussed. However, a more detailed description, including full sets of model parameters, can be found in the original publication [14].

The model assumes three different particle species: Fluorine, oxygen, and ions. The arrival directions of neutrals are assumed to follow the cosine law (2.5), while a power cosine distribution (2.8) is used for ions. For the description of the surface kinetics, coverages $\Theta_\mathrm{F}(\vec{x})$ and $\Theta_\mathrm{O}(\vec{x})$ are introduced, which describe the fraction of surface sites covered with fluorine and oxygen, respectively. The corresponding balance equations can be written in analogy to (2.34) as

$$\begin{aligned}
\sigma_\mathrm{Si}\frac{\Theta_\mathrm{F}}{dt}(\vec{x}) &= s_\mathrm{F}\left(1-\Theta_\mathrm{F}(\vec{x})-\Theta_\mathrm{O}(\vec{x})\right)F_\mathrm{F}(\vec{x}) - k\sigma_\mathrm{Si}\Theta_\mathrm{F}(\vec{x}) - 2\Theta_\mathrm{F}(\vec{x})Y_\mathrm{ie}^\mathrm{tot}(\vec{x}) \\
\sigma_\mathrm{Si}\frac{\Theta_\mathrm{O}}{dt}(\vec{x}) &= s_\mathrm{O}\left(1-\Theta_\mathrm{F}(\vec{x})-\Theta_\mathrm{O}(\vec{x})\right)F_\mathrm{O}(\vec{x}) - \beta\sigma_\mathrm{Si}\Theta_\mathrm{O}(\vec{x}) - \Theta_\mathrm{O}(\vec{x})Y_\mathrm{O}^\mathrm{tot}(\vec{x})
\end{aligned} \tag{6.1}$$

Here σ_Si is the surface site density of silicon. The first terms describe the adsorption of neutrals, which is proportional to the corresponding arriving fluxes, the sticking probabilities $s_\mathrm{F/O}$ on silicon, and the fraction of free surface sites $(1-\Theta_\mathrm{F}-\Theta_\mathrm{O})$. The second terms describe the loss of particles caused by chemical etching with rate k or by recombination with rate β, respectively. The third terms describe the removal due to ion-enhanced etching or sputtering. F_F and F_O are the total fluxes of neutrals on the surface. $Y_\mathrm{ie}^\mathrm{tot}$ is the total ion-enhanced etching rate and $Y_\mathrm{O}^\mathrm{tot}$ is the total oxygen sputter rate, which are both calculated using (2.27).

The surface velocity is equal to the total etch rate which is composed of three contributions, chemical etching, physical sputtering, and ion-enhanced etching (2.33)

$$V(\vec{x}) = -\frac{1}{\rho_\mathrm{Si}}\left(\frac{k\sigma_\mathrm{Si}\Theta_\mathrm{F}(\vec{x})}{4} + Y_\mathrm{ph}^\mathrm{tot}(\vec{x}) + \Theta_\mathrm{F}(\vec{x})Y_\mathrm{ie}^\mathrm{tot}(\vec{x})\right). \tag{6.2}$$

Here ρ_Si is the bulk density of silicon ($\rho_\mathrm{Si} = 5\times10^{28}\,\mathrm{m}^{-3}$).

The model fully incorporates specular reflexions of ions, which depend on the incident energy and direction. The reemitted direction follows the distribution given in (2.17). The functional dependence of the reemitted energy distribution on the incident energy is described in detail in [15].

The plasma etching process is first applied to a two layer structure, a silicon substrate covered by a 1.2 µm thick SiO_2 mask layer with a circular tapered hole. The diameter of the hole is 0.35 µm at the bottom and 0.4 µm at the top. The geometry is represented by two LSs resolved on a regular grid with lateral extensions 100×100 and a grid

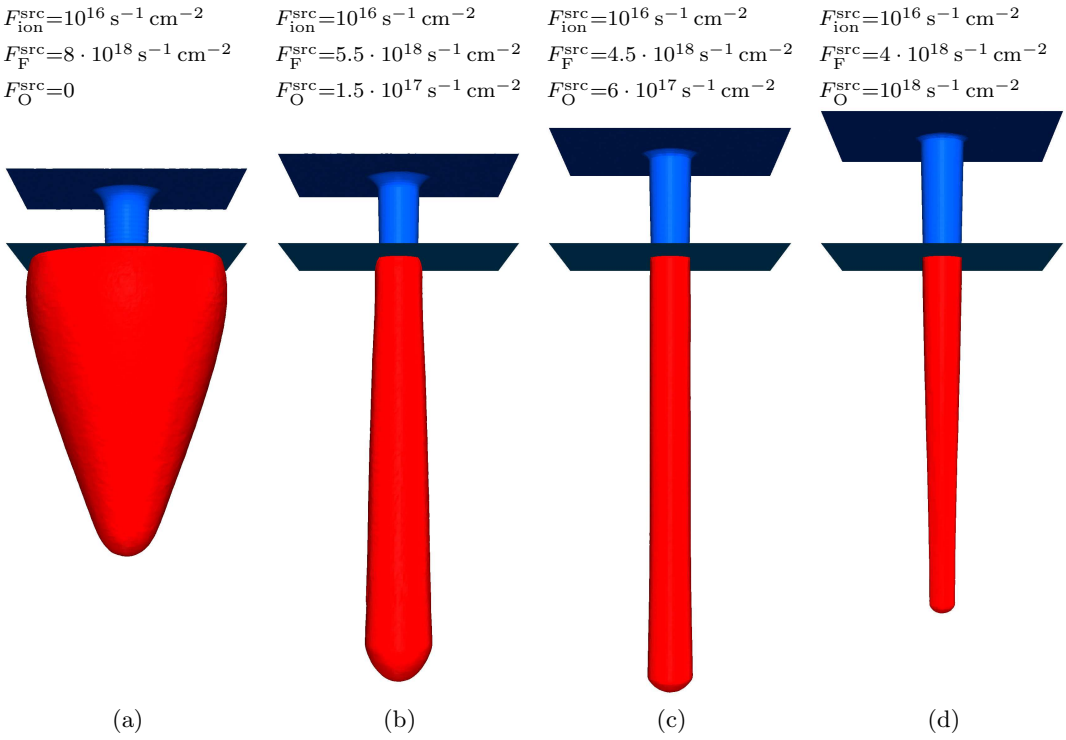
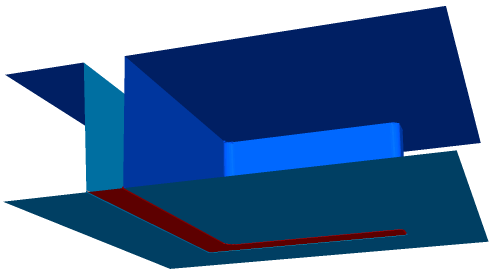


Figure 6.5: The final profiles after 150 s of etching in a SF_6/O_2 plasma for different gas compositions. The two level sets which are used to represent the two material regions, the mask and the substrate, are shown.

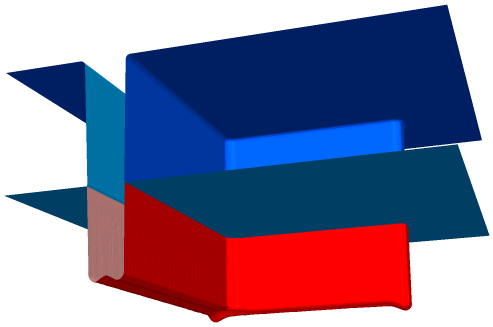
spacing of 20 nm. Reflective boundary conditions are assumed in both lateral grid directions.

Figure 6.5 shows the LS representations of the profiles after 150 s for different fluxes of oxygen and fluorine from the source. With increasing amount of oxygen, the etched profile gets more directional. The oxygen covers the sidewalls and prevents them from corrosion. Mask etching is also incorporated in these simulations. At every time step 5 million trajectories of each involved particle species are calculated. Due to the spatial subdivision and the use of modern quad-core processors the calculation time of one time step could be reduced to less than 15 s. About 2000 time steps are necessary to obtain the final profiles, resulting in a total computation time of approximately 8 h.

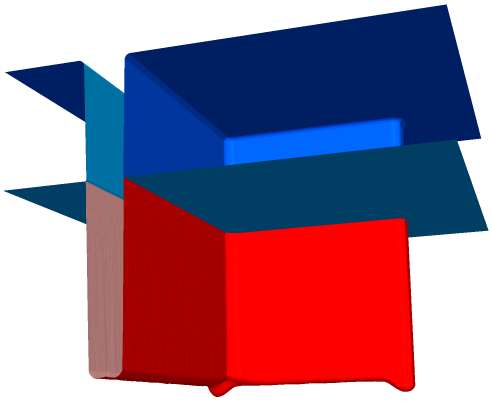
In a further example, a more complex structure (Figure 6.6a) is exposed to the same process parameters as listed in Figure 6.5c. The entire geometry is resolved on a grid with lateral extensions 200×200. Again, reflective boundary conditions and a grid spacing of 20 nm are used. The LS representations of the profile at different times are shown in Figure 6.6. For this simulation, due to the larger domain size, 20 million particle trajectories of each particle species were calculated at every time step. The full incorporation of specular reflections leads to micro-trenching at the trench bottom



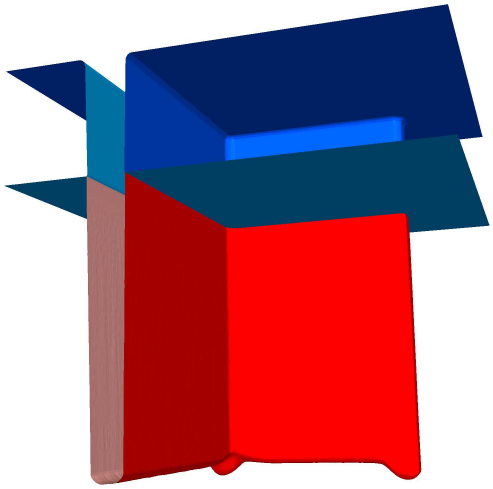
(a) The initial geometry. The mask has a thickness of 1.2 µm. The trench width is 0.35 µm at the bottom and 0.4 µm at the top.



(b) The profile after 25 s. Micro-trenching as a result of specular ion reflections can be clearly observed.



(c) The profile after 50 s.



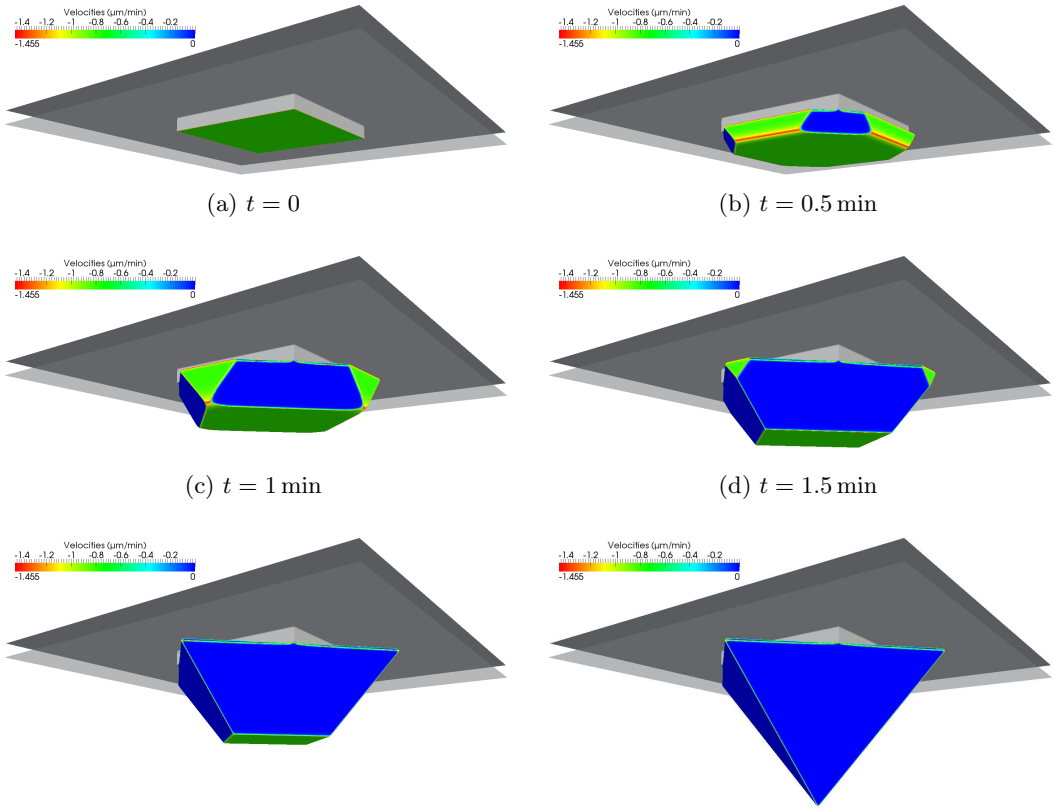
(d) The profile after 75 s.

Figure 6.6: Plasma etching simulation for a three-dimensional structure. The same parameters are used as for the simulation presented in Figure 6.5c.

edges (Figure 6.6b). For the same reason, the bend and the end of the trench are etched deeper.

6.3 Anisotropic Wet Etching

Another application is the simulation of anisotropic wet etching. In [96], a model for anisotropic wet etching of silicon with potassium hydroxide (KOH) is described. The material transport to the surface can be neglected for this process. The surface velocity only depends on the crystallographic orientation of the surface, thus on the local surface normal $\vec{n}$. It is assumed that $\vec{n}$ is given with respect to the crystallographic



(a) $t = 0$ (b) $t = 0.5\,\text{min}$

(c) $t = 1\,\text{min}$ (d) $t = 1.5\,\text{min}$

(e) $t = 2\,\text{min}$ (f) $t = 4\,\text{min}$

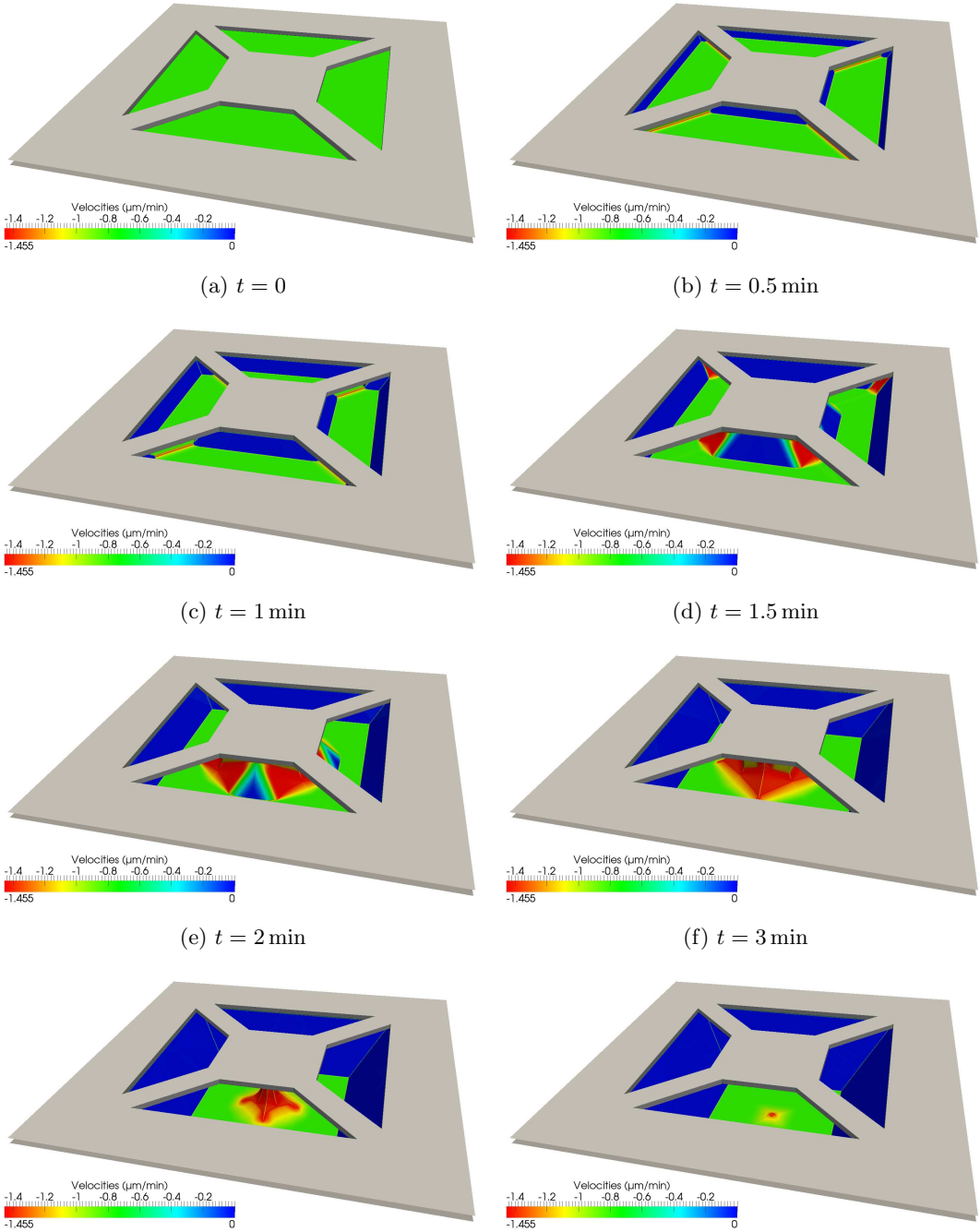
Figure 6.7: Anisotropic wet etching of a silicon substrate through a quadratic aperture. The lateral extensions of the simulation domain are $8\,\mu\text{m} \times 8\,\mu\text{m}$ and the grid resolution is $\Delta x = 10\,\text{nm}$. The simulation boundaries are aligned to $\langle 100 \rangle$ directions.

directions, [100], [010], and [001]. Utilizing the crystallographic symmetries of silicon, it is possible to only consider the case $1 \geq n_3 \geq n_2 \geq n_1 \geq 0$. This allows the surface velocity to be approximated by

$$V(\vec{x}) = \begin{cases} \frac{-R_{100}(n_1 - n_2 - 2n_3) - R_{110}(n_2 - n_3) - 3R_{311}n_3}{n_1} & \text{if } n_1 - n_2 - 2n_3 > 0, \\ \frac{R_{111}(n_1 - n_2 - 2n_3) - 2R_{110}(n_2 - n_3) - 3R_{311}(n_1 - n_2)}{2n_1} & \text{else.} \end{cases} \tag{6.3}$$

Here the model parameters R_{100}, R_{110}, R_{111}, and R_{311} are the etch rates for the crystallographic directions $\langle 100 \rangle$, $\langle 110 \rangle$, $\langle 111 \rangle$, and $\langle 311 \rangle$, respectively. (6.3) interpolates these rates for all other directions in between. For a KOH concentration of 30 % and a temperature of 70 °C, the measured values of these rates are $R_{100} = 0.797\,\mu\text{m}\,\text{min}^{-1}$, $R_{110} = 1.455\,\mu\text{m}\,\text{min}^{-1}$, $R_{111} = 0.005\,\mu\text{m}\,\text{min}^{-1}$, and $R_{311} = 1.436\,\mu\text{m}\,\text{min}^{-1}$ [103].

Inserting these numerical values into (6.3) leads to a non-convex Hamiltonian for the LS equation. Hence, to obtain a stable solution, a non-convex scheme, such as the Lax-Friedrichs scheme, must be applied (see Section 3.2.2). For the simulations, presented



(a) $t = 0$ (b) $t = 0.5\,\text{min}$

(c) $t = 1\,\text{min}$ (d) $t = 1.5\,\text{min}$

(e) $t = 2\,\text{min}$ (f) $t = 3\,\text{min}$

(g) $t = 4\,\text{min}$ (h) $t = 5\,\text{min}$

Figure 6.8: The LS representations at different times for an anisotropic wet etching process. The lateral domain boundaries are aligned to $\langle 110 \rangle$ directions. The structure size is $20\,\mu\text{m} \times 20\,\mu\text{m}$.

in the following, the dissipation coefficients α_i in (3.12) are all set to $2.8\,\mu\mathrm{m\,min}^{-1}$. This value has been found through variation and selection of the value that gives the most accurate simulation results.

Some examples presented in [96] are reproduced by the following simulations. First, anisotropic wet etching of a silicon substrate through a mask with a quadratic hole is simulated. Due to the optimized LS framework, it is possible to use a very fine grid with a grid spacing of just 10 nm. Since the extensions of the structure are $8\,\mu\mathrm{m} \times 8\,\mu\mathrm{m}$, the lateral grid extensions are 800×800. Two LS functions are used to represent the mask and the etch front. Figure 6.7 shows the LS representation at different times. The total calculation time was about 100 min on two cores of an AMD Opteron 8222 SE processor (3 GHz). If $C_{\mathrm{CFL}} = 0.1$ is used for the CFL condition, approximately 3900 time steps are necessary. The total memory consumption did not exceed 360 MB.

The results of a second simulation are shown Figure 6.8. There, a more complex mask structure is assumed with size $20\,\mu\mathrm{m} \times 20\,\mu\mathrm{m}$. This leads to a grid with lateral extensions of 2000×2000. Thanks to the H-RLE data structure, the total memory consumption was always below 1.2 GB. For comparison, the usage of a full grid with dimensions $2000 \times 2000 \times 500$ would require at least 30 GB for just storing the two LS functions. Using again $C_{\mathrm{CFL}} = 0.1$, the total calculation time for the required 12 400 time steps was about 34 h on two cores of an AMD Opteron 8222 SE processor. The final profiles of both simulations are in good agreement with those reported in [96].

6.4 Bosch Process

The Bosch process has already been mentioned in Section 4.7.2, when demonstrating the multi-LS framework. A sequence of passivation and etching cycles can be used for high aspect ratio etching. In each cycle a chemically inert polymer layer is uniformly deposited using fluorocarbon gases. This passivation layer prevents the sidewalls from being attacked in the subsequent etching step (compare Figure 4.15).

By feeding a high frequency plasma with etch gases like SF_6, CF_4, or NF_3, a superposition of physical (directional) and chemical (isotropic) etching is obtained. This leads to a faster removal of the passivation layer at the bottom of the trench compared to the sidewalls due to the additional sputtering of the directional ions. After uncovering the substrate at the bottom, chemical etching is dominant. Hence, in simple terms, in each cycle an isotropic etching process is initiated at the bottom of the trench. After many iterations, profiles with high aspect ratios can be obtained. For optimal processing the passivation and etching cycle times must be balanced. If the deposited passivation layer is too thin, the process time for the etching cycle must be smaller to avoid the corrosion of the sidewalls, increasing the number of required iterations. If the layer is too thick, the etching duration must be increased, resulting in a longer

total process time. The choice of the process times also has an influence on the undulation of the sidewalls caused by the two-phase procedure. Computer simulations help to study parameter variations in order to optimize the process.

A two-dimensional simulator using a cell-based method for surface evolution was presented in [137, 138]. Therein a simplified model for the particle transport is used. Etching is modeled by an isotropic etch rate superposed by a directional term which is proportional to the incident ion flux. For the passivation cycle a perfect conformal deposition is assumed, which is equivalent to a constant surface velocity. However, this model is not able to describe the lag effect [35] appropriately. Therefore, a geometric shape factor was introduced [128] to account for different trench widths.

A simulation with a more sophisticated transport model is presented in [63], where different sticking probabilities and higher order reemissions of neutral particles are incorporated using a ballistic transport and reaction model [20, 21]. Since the transport of neutrals to the surface is taken into account, the lag effect is inherently incorporated. The surface evolution is calculated using the LS method, while the particle transport is computed using the conventional approach [62]. The MC based calculation of the particle transport was first applied to Bosch process simulation in combination with a cell-based method for surface evolution in [116]. Many particle trajectories and their surface reactions are calculated to determine the surface rates.

Three-dimensional simulations of the Bosch process were recently reported in [43] and [125]. Both use simplified transport models and do not incorporate higher order reemissions of neutrals. Instead, a uniform surface rate is assumed. The particle transport is calculated using conventional integral methods. For surface evolution, a cell-based method and the LS method are used, respectively.

The numerical methods presented in the previous chapters allow an efficient solution of the more realistic model described in [63] even in three dimensions. The model assumes linear surface reactions for the passivation and the etching cycle (see Section 2.3.1)

$$V(\vec{x}) = \alpha F_{\text{ion}}(\vec{x}) + \beta F_{\text{neu}}(\vec{x}). \tag{6.4}$$

The coefficients α and β are model parameters, and in case of etching they depend on the material on the surface. The sticking probabilities are assumed to be constant for neutrals. The arriving directions of neutrals follow a cosine distribution (2.5). A power cosine distribution (2.8) is used for ions. The energy distribution of ions is not considered. The numeric values of all parameters for the passivation and the etching cycle, which are used for all simulations, are listed in Table 6.1 and Table 6.2. Contrary to [63], mask etching is also considered by assuming a mask/substrate etch selectivity of 1:20. Hence, the coefficients α and β for the mask are adjusted accordingly.

Parameter	Value	Description
$F_{\text{neu}}^{\text{src}}$	$2 \times 10^{18}\,\text{cm}^{-2}\,\text{s}^{-1}$	neutral flux
$F_{\text{ion}}^{\text{src}}$	$3.125 \times 10^{15}\,\text{cm}^{-2}\,\text{s}^{-1}$	ion flux
ν	820	exponent for ions in (2.8)
α	$10\,\text{Å}^3$	coefficient in (6.4)
β	$0.5\,\text{Å}^3$	coefficient in (6.4)
s	0.1	sticking probability for neutrals

Table 6.1: The numeric values of the parameters used for the passivation cycle of the Bosch process.

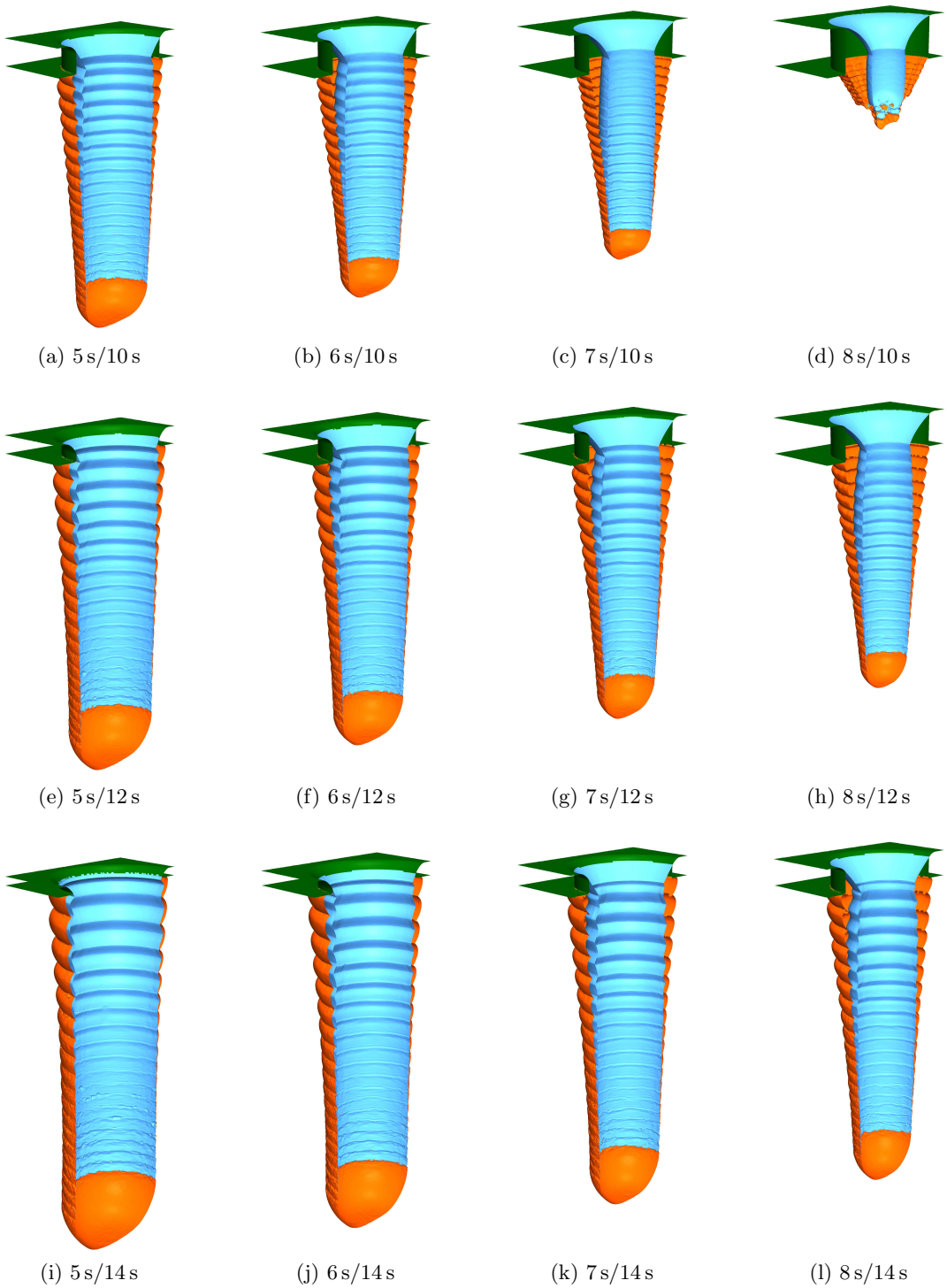
Parameter	Value	Description
$F_{\text{neu}}^{\text{src}}$	$10^{19}\,\text{cm}^{-2}\,\text{s}^{-1}$	neutral flux
$F_{\text{ion}}^{\text{src}}$	$4.375 \times 10^{15}\,\text{cm}^{-2}\,\text{s}^{-1}$	ion flux
ν	820	exponent used for ions in (2.8)
α_{polymer}	$-125\,\text{Å}^3$	coefficient in (6.4) for polymer
$\alpha_{\text{substrate}}$	$-270\,\text{Å}^3$	coefficient in (6.4) for substrate
α_{mask}	$-13.5\,\text{Å}^3$	coefficient in (6.4) for mask
β_{polymer}	$-0.03\,\text{Å}^3$	coefficient in (6.4) for polymer
$\beta_{\text{substrate}}$	$-0.9\,\text{Å}^3$	coefficient in (6.4) for substrate
β_{mask}	$-0.045\,\text{Å}^3$	coefficient in (6.4) for mask
s_{polymer}	0.1	sticking probability for neutrals on polymer
$s_{\text{substrate}}$	0.2	sticking probability for neutrals on substrate
s_{mask}	0.2	sticking probability for neutrals on mask

Table 6.2: The numeric values of the parameters used for the etching cycle of the Bosch process.

6.4.1 Process Time Variations

The effect of different passivation and etching cycle durations is studied on a structure composed of a substrate and a 1 µm thick mask which has a cylindrical hole with 2.5 µm diameter. Despite the rotational symmetry, this problem cannot be straightforwardly reduced to two dimensions. The introduction of cylindrical coordinates leads to non-linear particle trajectories, which makes the determination of the visibilities between two points on the surface much more difficult. For convex holes, where all points are visible between each other, the solution of the transport equation for cylindrical coordinates using the conventional approach was demonstrated in [62]. However, due to the rippled, non-convex side walls of the hole, which evolve during the Bosch process, this method cannot be applied.

In three dimensions, the simulation domain can be reduced to a quarter due to the reflective boundary conditions and the twofold reflection symmetry of the hole. However, to prove the symmetry of the solution and to avoid reflections, the final visualizations



(a) 5 s/10 s (b) 6 s/10 s (c) 7 s/10 s (d) 8 s/10 s

(e) 5 s/12 s (f) 6 s/12 s (g) 7 s/12 s (h) 8 s/12 s

(i) 5 s/14 s (j) 6 s/14 s (k) 7 s/14 s (l) 8 s/14 s

Figure 6.9: The final profiles after 20 cycles for different combinations of deposition/etching process times.

are generated with a process simulation on half of the domain, which is discretized using a grid with lateral extensions 140×70. The grid spacing is 25 nm for all simulations. 1.96 million particles are launched from the source plane at every time step.

The final profiles, after 20 cycles with different process times for deposition (5 s, 6 s, 7 s, and 8 s) and etching (10 s, 12 s, and 14 s) are given in Figure 6.9. The results show the influence of the process time on the depths of the holes, tilt angles of the side walls, and the resulting polymer layers. Since mask etching is also incorporated, its final thickness can be studied. Such simulations can help find the optimal process parameters.

6.4.2 Lag Effect

The influence of the hole diameter on the final profile is further investigated. A Bosch process with 6 s passivation followed by 12 s etching cycles is applied on a 1 µm thick perforated mask with cylindrical holes of varying diameters (0.5 µm, 1 µm, 1.5 µm, 2 µm, and 2.5 µm).

The simulation domain is resolved on a grid with extensions 500×140. 12.5 million particles are simulated for each time step. Using 8 cores of AMD Opteron 8222 SE processors (3 GHz) the total computation time is approximately 2 days. Approximately 6500 time steps are necessary to simulate all 20 cycles of the Bosch process.

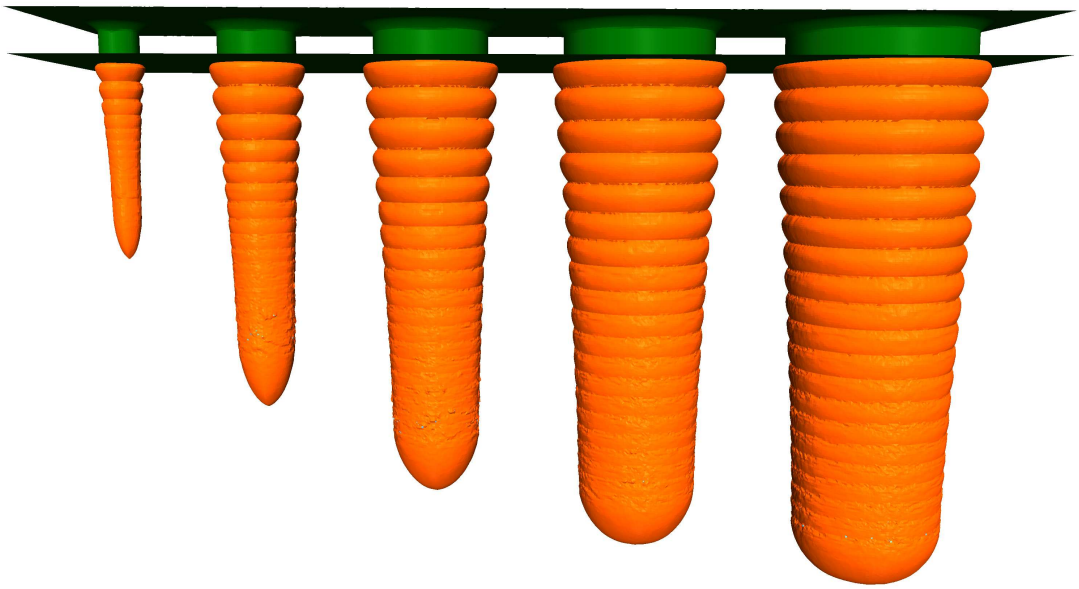


Figure 6.10: Deep reactive ion etching of holes with varying diameters (0.5 µm, 1 µm, 1.5 µm, 2 µm, and 2.5 µm). The lag effect is the reason for the different depths. The structure is resolved on a grid with lateral extensions 500×140.

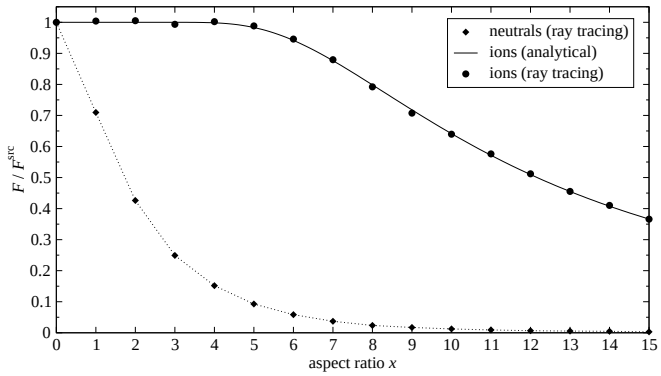


Figure 6.11: The characteristic dependence of the neutral and ion fluxes at the bottom center on the aspect ratio.

The calculation time for one time step is 27 s on average. The runtimes increase continuously during the entire simulation due to the increasing depths of the holes and the increasing surface area. Figure 6.10 shows the final profile after 20 cycles. The different etching depths due to the lag effect can be observed. With increasing aspect ratio, the effective etching rate decreases.

To analyze the reason behind the lag effect in more detail, the ion and neutral fluxes are calculated at the bottom center of idealized cylindrical holes for various aspect ratios x, which is a ratio of the depth and the hole diameter. The ion fluxes obtained by ray tracing are in very good agreement with those calculated analytically (Figure 6.11). The analytical expression

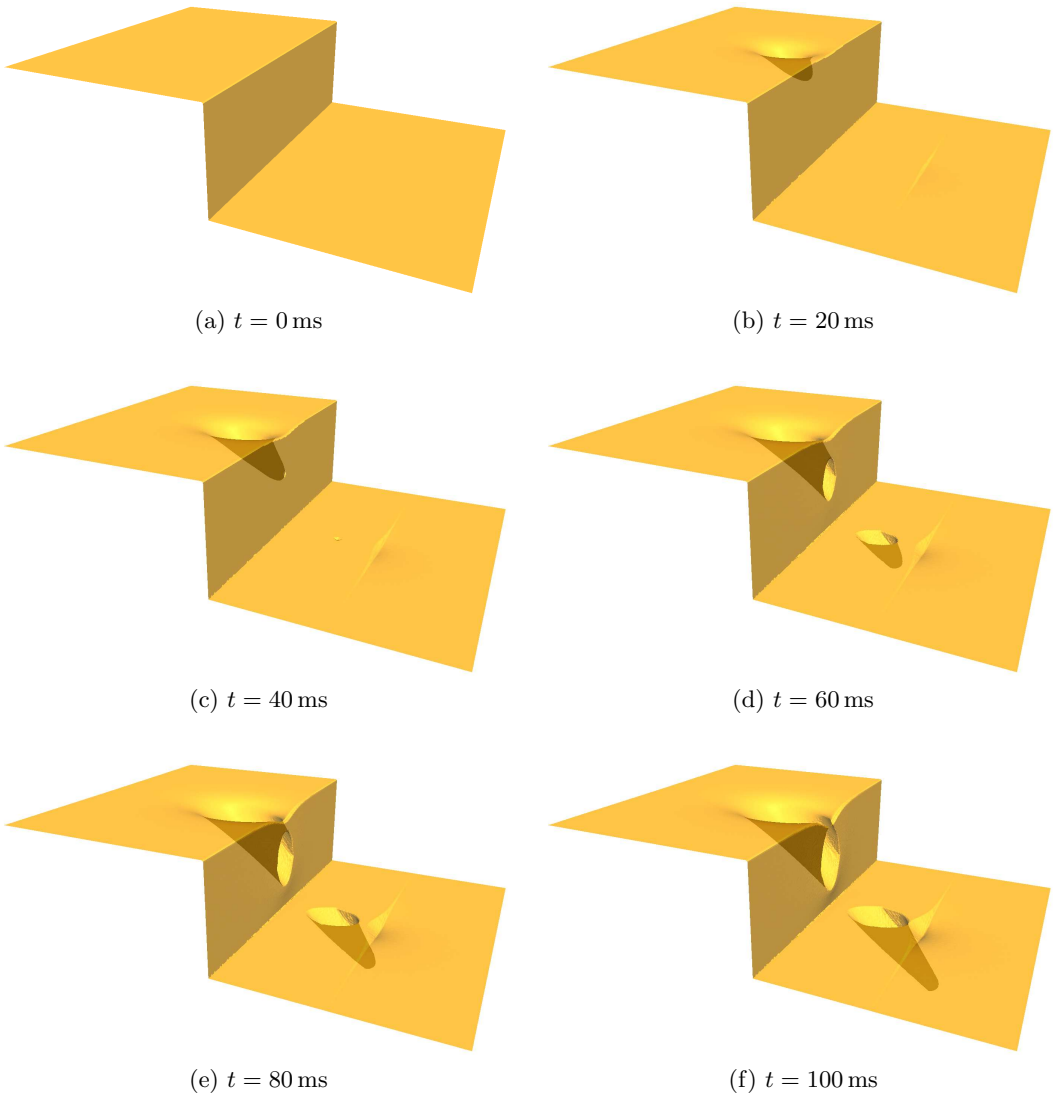
$$F_{\text{ion}} = F_{\text{ion}}^{\text{src}} \cdot \left(1 - \left(\frac{2x}{\sqrt{1+4x^2}}\right)^{\nu+1}\right) \tag{6.5}$$

can be derived from (5.10) by integrating over the open solid angle of the cylindrical hole. For the calculation of the neutral flux, the sticking coefficient is set to 0.1, which corresponds to the sticking probability of neutrals on the passivation layer, as used in this model. The results show that the aspect ratio affects the neutral flux much more than the directional ion flux. With increasing depth the hole surface area increases, leading to a smaller fraction of particles which remain sticking at the bottom and not at the sidewalls.

According to (6.4) and Table 6.1 the neutral flux is the main contributor to the deposition rate of the passivation layer. Hence, by increasing the aspect ratio, the thickness of the deposited passivation layer decreases due to the smaller neutral fluxes. However, unlike the neutral flux, the ion flux is not reduced significantly. As a consequence, the passivation layer is etched through much faster. The ion flux countervails the lag effect, because the substrate is attacked earlier in the etching cycle for larger aspect ratios. However, this head start is more than compensated by the larger substrate etch rate for smaller aspect ratios, due to the larger neutral fluxes.

6.5 Focused Ion Beam Processing

Another application of the presented simulation techniques are FIB simulations. So far, full three-dimensional simulations have been presented in [55] and [57] using cell- and segment-based methods for surface evolution, respectively. The application of the LS method for FIB simulations was first demonstrated in [58]; however, only for the two-dimensional case. In this section, three-dimensional FIB simulations using the LS method and ray tracing are presented.

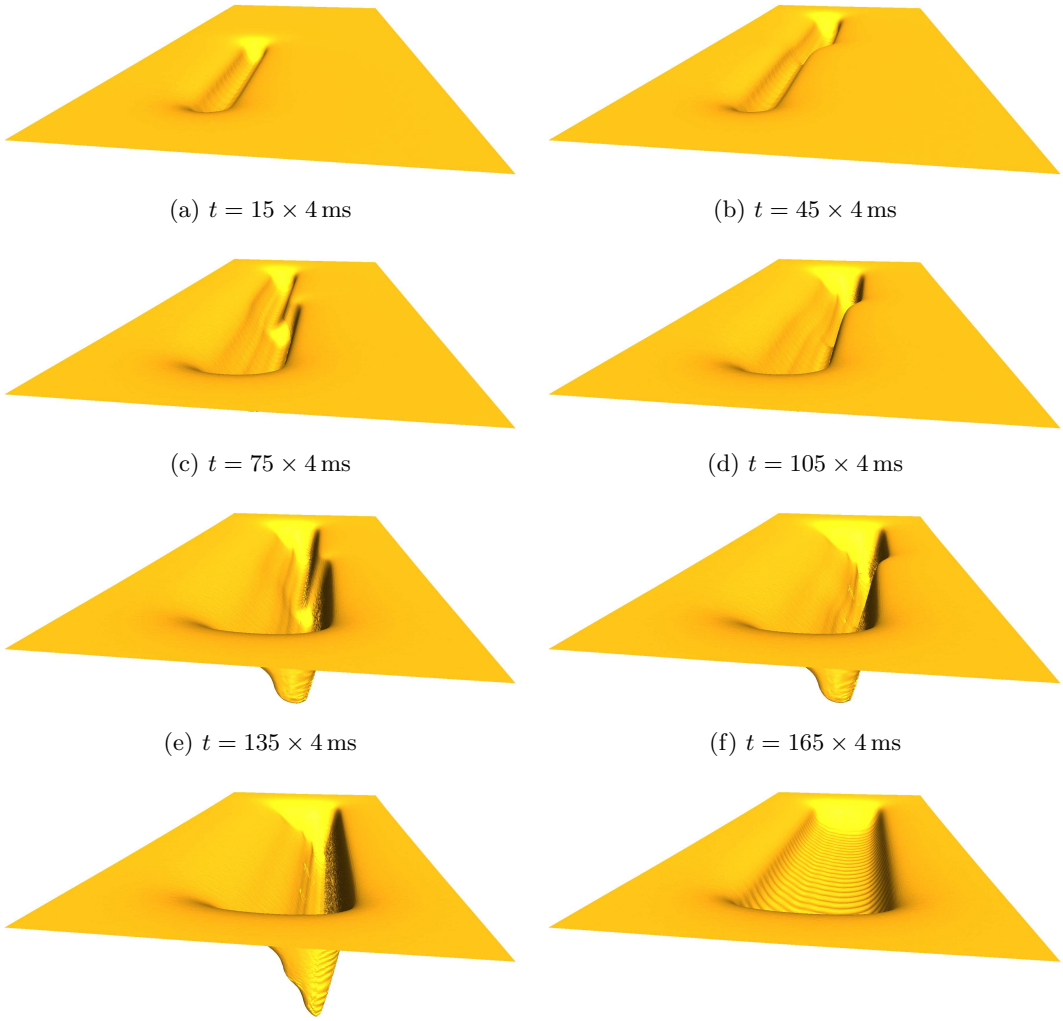


(a) $t = 0\,\text{ms}$ (b) $t = 20\,\text{ms}$

(c) $t = 40\,\text{ms}$ (d) $t = 60\,\text{ms}$

(e) $t = 80\,\text{ms}$ (f) $t = 100\,\text{ms}$

Figure 6.12: Ion beam milling of a step structure for an incident angle of 45°.



(a) $t = 15 \times 4\,\text{ms}$ (b) $t = 45 \times 4\,\text{ms}$

(c) $t = 75 \times 4\,\text{ms}$ (d) $t = 105 \times 4\,\text{ms}$

(e) $t = 135 \times 4\,\text{ms}$ (f) $t = 165 \times 4\,\text{ms}$

(g) $t = 180 \times 4\,\text{ms}$ (h) $t = 720 \times 1\,\text{ms}$

Figure 6.13: (a)-(g) Serpentine scan of 30×6 pixels with dwell time of 4 ms. (h) Four passes of a serpentine scan with dwell time of 1 ms.

The same model is used as described in [57], which also considers redeposition of sputtered material. Sputtered particles are assumed to remain sticking with a probability $s = 1$. The surface velocity is given by the difference of the sputter rate and the deposition rate of redeposited particles. Here, the sputter rate is modeled using Yamamura's formula (2.21). There, the constants C_1 and C_2 are chosen in such a way that the maximum sputter rate is 20 particles per ion for an incident angle of 82°. For normal incidence a sputter rate of 2.5 is assumed. The directions of sputtered particles are described by a cosine distribution (2.19). The beam profile is modeled using a normal distribution (2.11).

The first example (Figure 6.12) shows the effect of an ion beam with diameter 50 nm (FWHM) and inclined incidence with angle 45° on a step structure. The LS method is able to describe the appearing topographic changes without any problems. The beam current is set to 50 pA and the bulk density is assumed to be that of silicon ($\rho_{\mathrm{Si}} = 5 \times 10^{28}\,\mathrm{m}^{-3}$). The grid spacing is set to 2 nm. One million ions are launched at every time step in order to calculate the surface velocities.

As a second example, the time evolution of a plain surface, which is processed by a normal incident FIB as described in [57], is calculated. The beam is moved over 30×6 pixels using a serpentine scan strategy. The overlap of two neighboring pixels is assumed to be 50 %, which means that the distance between their midpoints is 50 % of the beam diameter, thus 25 nm. Figure 6.13 shows the results for two different processing schemes: a single pass with a dwell time of 4 ms at every pixel and four passes with a dwell time of 1 ms.

7 Summary and Outlook

In this work many new numerical techniques and algorithms for topography simulation, especially for large three-dimensional geometries, have been presented. The combination of modern LS techniques, such as the sparse field method and the H-RLE data structure, lead to a fast optimal scaling surface evolution algorithm. Novel iterators for accessing the H-RLE data structure have been implemented. They enable fast serial processing of the H-RLE data structure with proper incorporation of boundary conditions. The developed iterators are used for the implementation of the sparse field method and for the realization of Boolean operations. Furthermore, efficient algorithms for testing unidirectional visibility and connectivity, which are useful for certain process models, have been described. In addition, a new way to handle multiple material regions, including thin layers, using a multi-LS description has been presented. The LS framework was further enhanced by adapting the H-RLE data structure for parallelization on shared-memory machines.

General surface kinetics models require the calculation of the particle transport in order to obtain surface velocities. Since ballistic particle transport can be assumed for many processes, ray tracing techniques, such as spatial subdivision, can be applied. A new data structure using neighbor links arrays was suggested, which appeared to be very suitable and efficient for topography simulation. Furthermore, ray tracing was directly applied to the implicit LS representation of the surface. Rates are directly calculated for surface points using tangential disks. Hence, no explicit surface representation is required, which saves memory and computation time.

Contrary to the conventional direct integration approach used for surface rate calculation, ray tracing enables the incorporation of higher order reemissions as well as effects which depend on the direction and energy of incident particles. At the same time, a reduced computational complexity is obtained and parallelization is straightforward. Furthermore, ray tracing allows the definition of a simple interface, which enables an easy implementation of new process models. Together with the LS framework, a powerful and efficient topography simulator was created, which can be used for a large variety of processes. As a demonstration, a selection of process models reported in the literature has been implemented and used for various three-dimensional applications.

For future work, it might be interesting to develop methods and algorithms, which are able to convert back the LS representation into a volume mesh of high quality, which can then be used for subsequent simulation of process steps requiring a volume mesh,

e.g. diffusion. A promising approach for meshing an implicit surface is described in [18, 82, 129]. The idea is to start from a regular tetrahedral mesh which is refined near the surface. Tetrahedra outside of the surface are removed, and the remaining tetrahedral mesh is adapted to the surface by a relaxation procedure.

As already mentioned in Section 2.2.2, some processes might require the incorporation of electrostatic interactions. Ion incidence may charge the surface, which leads to an electrostatic field that deflects ion trajectories. The effect of surface charging has already been fully incorporated in two-dimensional MC simulations [104]. However, the accuracy of three-dimensional simulations is still limited by the high computational costs [94, 97]. To speed up the calculation, it might be possible to extend the presented ray tracing techniques as follows: First, the electric field is calculated at all corners of each subbox of the spatial subdivision. This enables a multi-linear interpolation of the field within each subbox. Then, the ion trajectory through a subbox can be calculated using a standard leapfrog method [92]. If necessary, the spatial subdivision needs to be refined at regions with large field gradients.

Appendix A

Line–Triangle Intersection

For the initialization of the LS function, triangles must be checked for possible intersections with grid lines (see Section 4.1.1). In the following, a robust line–triangle intersection test is presented. If the line intersects the triangle, the intersection point is also calculated. The following approach ensures that, if a line close to an edge fails the intersection test due to numerical errors, the test will succeed for another triangle, which is adjacent to the same edge. Hence, if a surface is given as a triangulation, cases, where the line fails all intersection tests, although it intersects the surface, are avoided.

A triangle with vertices $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$ is intersected by a line defined by point $\vec{a}$ and unit vector $\vec{\omega}$, if the (signed) areas

$$A_i := \frac{1}{2}\left((\vec{x}_{i+1} - \vec{a}) \times (\vec{x}_{i+2} - \vec{a})\right) \cdot \vec{\omega} = \frac{1}{2}\det(\vec{x}_{i+1} - \vec{a}, \vec{x}_{i+2} - \vec{a}, \vec{\omega}) \tag{A.1}$$

are either all non-negative or all non-positive. The sign is given by the triangle orientation with respect to $\vec{\omega}$. All the index summations in (A.1) are modulo 3 plus 1. A_i corresponds to the area of the triangle spanned by the vectors $\vec{x}_{i+1}$, $\vec{x}_{i+2}$, and $\vec{a}$, if it is projected onto a plane perpendicular to $\vec{\omega}$ (see Figure A.1). If all A_i are equal to zero, the grid line is in the same plane as the triangle, or the triangle is degenerated. In both cases the line is considered to not intersect the triangle.

To guarantee that lines close to an edge succeed the intersection test for any of the two adjacent triangles, the numerical evaluation of (A.1) must preserve the anticommutativity with respect to $\vec{x}_{i+1}$ and $\vec{x}_{i+2}$. However, due to numerical errors this is usually not the case. For example, the numerical result of $a \cdot b - c \cdot d$ is not necessarily the same as that of $c \cdot d - a \cdot b$ with opposite sign [100]. To overcome this problem, the two vectors $\vec{x}_{i+1}$ and $\vec{x}_{i+2}$ in (A.1) are compared using an order relation, such as lexicographical comparison. If $\vec{x}_{i+1}$ is larger than $\vec{x}_{i+2}$, the two vectors are swapped, and the final result of (A.1) is inverted. This technique ensures the anticommutativity in (A.1).

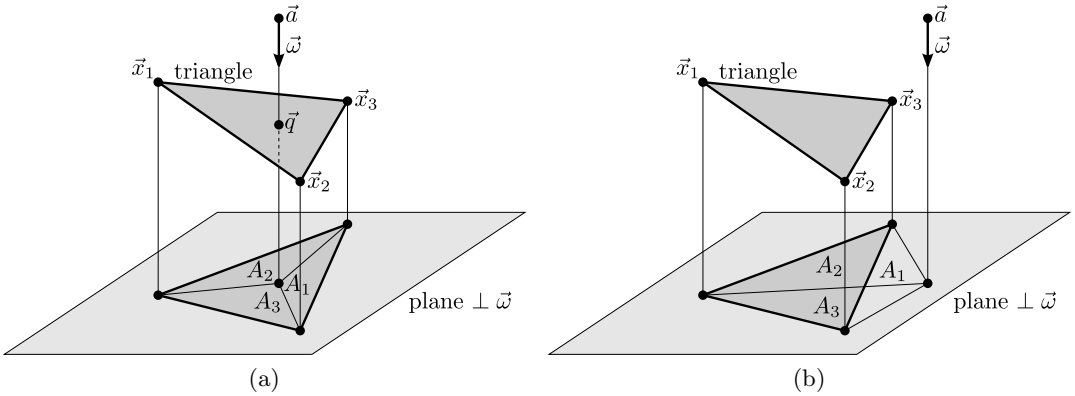


Figure A.1: The line defined by the point $\vec{a}$ and the unit vector $\vec{\omega}$ intersects the triangle given by $\vec{x}_1$, $\vec{x}_2$, and $\vec{x}_3$, if the signed areas A_1, A_2, and A_3 have the same sign. (a) $A_1 < 0$, $A_2 < 0$, and $A_3 < 0$, which implies that the line intersects the triangle. (b) No intersection, since $A_1 > 0$, $A_2 < 0$, and $A_3 < 0$.

Once the areas A_i are determined and the intersection test is positive, the intersection point $\vec{q}$ can be obtained by

$$\vec{q} = \frac{\sum_{i=1}^{3} A_i \cdot \vec{x}_i}{A} = \sum_{i=1}^{3} \frac{A_i}{A} \cdot \vec{x}_i \quad \text{with } A := \sum_{i=1}^{3} A_i. \tag{A.2}$$

As defined earlier, an intersection only occurs, if all A_i are non-negative or non-positive, while not all are equal to zero. Therefore, the denominator A in (A.2) never vanishes. To avoid potential overflows the second variant for the calculation of the intersection point is preferable. There, the coefficients $\frac{A_i}{A}$ are always in the range $[0, 1]$ and the calculation of the intersection point is safe.

Appendix B

Ray–Isosurface Intersection

It is assumed that the surface is represented as the zero LS of function $\Phi(\vec{x})$ and a ray with direction $\vec{\omega}$ passes through point $\vec{a}$

$$\vec{x}(t) = \vec{a} + t \cdot \vec{\omega}. \tag{B.1}$$

In order to find the intersection with the surface, the following equation must be solved

$$\Phi(\vec{x}(t)) = 0. \tag{B.2}$$

It is essential for ray tracing, that this equation can be solved with as few numerical operations as possible. Optimized algorithms are presented in the following, which are superior to those reported in the literature [69, 74, 88, 119].

The LS function Φ is usually multi-linearly interpolated within a grid cell. The index vector of a grid cell is defined to be equal to the lower bound indices of all its corner grid points. For an arbitrary number of dimensions D, the multi-linear interpolation formula within the grid cell with index vector $\vec{p}$ can be expressed as

$$\Phi(\vec{p}+\vec{x}) \approx \sum_{\vec{\alpha}\in\{0,1\}^D} \Phi(\vec{p}+\vec{\alpha}) \prod_{i=1}^{D} \sum_{\beta_i=\alpha_i}^{1} (-1)^{\alpha_i+\beta_i} x_i^{\beta_i}, \quad \forall i : 0 \le x_i \le 1. \tag{B.3}$$

This expression can be expanded to a multi-variate polynomial

$$\Phi(\vec{p}+\vec{x}) \approx \sum_{\vec{\beta}\in\{0,1\}^D} \left(\prod_{i=1}^{D} x_i^{\beta_i} \right) \rho_{\vec{\beta}}(\vec{p}), \tag{B.4}$$

where $\rho_{\vec{\beta}}(\vec{p})$ are its coefficients. They can be obtained by rewriting (B.3) as

$$\Phi(\vec{p}+\vec{x}) \approx \sum_{\vec{\alpha}\in\{0,1\}^D} \Phi(\vec{p}+\vec{\alpha}) \sum_{\beta_1=\alpha_1}^{1} \cdots \sum_{\beta_D=\alpha_D}^{1} \prod_{i=1}^{D} (-1)^{\alpha_i+\beta_i} x_i^{\beta_i} \tag{B.5}$$

and further as

$$\Phi(\vec{p}+\vec{x}) \approx \sum_{\vec{\beta}\in\{0,1\}^D} \left(\prod_{i=1}^{D} x_i^{\beta_i} \right) \sum_{\alpha_1=\beta_1}^{1} \cdots \sum_{\alpha_D=\beta_D}^{1} \Phi(\vec{p}+\vec{\alpha}) \prod_{i=1}^{D} (-1)^{\alpha_i+\beta_i}. \tag{B.6}$$

Hence, the coefficients can be calculated as

$$\rho_{\vec{\beta}}(\vec{p}) = \sum_{\alpha_1=\beta_1}^{1} \cdots \sum_{\alpha_D=\beta_D}^{1} \Phi(\vec{p}+\vec{\alpha}) \prod_{i=1}^{D} (-1)^{\alpha_i+\beta_i}. \tag{B.7}$$

If the coefficients are calculated in a straightforward manner, $3^D - 2^D$ additions or subtractions are needed [69, 119]. However, the number of numerical operations can be reduced to $2^{D-1}D$ subtractions as demonstrated in Algorithm B.1. The algorithm is realized using template metaprogramming [2] which allows for the elimination of all recursive function calls at compile time. An array of size 2^D, which contains the LS values of all corner grid points in lexicographical order (with reversed significance), is passed to the static member function. After the function call the same array holds the coefficients $\rho_{\vec{\beta}}(\vec{p})$ in lexicographical order (again with reversed significance) with respect to $\vec{\beta}$.

Algorithm B.1 Calculation of the interpolation coefficients from the LS values using template metaprogramming in C++.

```
template<int D> struct transform
{
    static void execute(double x[])
    {
        enum {t=1<<(D-1)}; //t = 2^(D-1)
        transform<D-1>::execute(x);
        transform<D-1>::execute(x+t);
        for (int i=0;i<t;++i) x[i+t]-=x[i];
    }
};

template<> struct transform<0> // class template specialization
{
    static void execute(double x[]) {}
};
```

B.1 Setup of the Polynomial

Inserting the ray parameterization (B.1) into (B.4) leads to a polynomial of order D

$$\sum_{i=0}^{D} C_k t^k. \tag{B.8}$$

The calculation of the coefficients C_k described in the literature [69, 74, 88, 119] is not optimal in terms of the number of required multiplications. Therefore, optimized algorithms, Algorithm B.2 and Algorithm B.3, have been developed for the

two- and three-dimensional cases, respectively. There, ρ_k denotes the k-th element of the multi-linear polynomial coefficients $\rho_{\vec{\beta}}$, if they are sorted in lexicographical order. For example, ρ_3 corresponds to $\rho_{(1,1,0)}$ in the three-dimensional case.

Algorithm B.2 Calculation of the coefficients for the two-dimensional case.

$C_2 \leftarrow \omega_1 \cdot \rho_3$
$C_0 \leftarrow a_1 \cdot \rho_3 + \rho_2$
$C_1 \leftarrow a_2 \cdot C_2 + \omega_2 \cdot C_0 + \omega_1 \cdot \rho_1$
$C_0 \leftarrow a_2 \cdot C_0 + a_1 \cdot \rho_1 + \rho_0$
$C_2 \leftarrow \omega_2 \cdot C_2$

Algorithm B.3 Calculation of the coefficients for the three-dimensional case.

$C_1 \leftarrow a_2 \cdot \rho_7 + \rho_5$
$C_0 \leftarrow C_1 \cdot a_1 + a_2 \cdot \rho_6 + \rho_4$
$C_1 \leftarrow C_1 \cdot \omega_1 + \omega_2 \cdot (a_1 \cdot \rho_7 + \rho_6)$
$C_2 \leftarrow \omega_1 \cdot \omega_2$
$C_3 \leftarrow C_2 \cdot \omega_3 \cdot \rho_7$
$C_2 \leftarrow C_2 \cdot (a_3 \cdot \rho_7 + \rho_3) + C_1 \cdot \omega_3$
$T \leftarrow a_1 \cdot \rho_3 + \rho_2$ // T is a temporary variable
$C_1 \leftarrow C_1 \cdot a_3 + C_0 \cdot \omega_3 + (a_2 \cdot \rho_3 + \rho_1) \cdot \omega_1 + T \cdot \omega_2$
$C_0 \leftarrow C_0 \cdot a_3 + T \cdot a_2 + a_1 \cdot \rho_1 + \rho_0$

B.2 Root Finding

Once the polynomial is set up for a grid cell, it can be tested for any ray–surface intersection within its interior. Let $[t_{\min}, t_{\max}]$ be the set of parameter values, for which the ray lies inside the corresponding grid cell. If there exists any real root in this interval, the ray intersects the surface. In this case the smallest real root, which corresponds to the first surface intersection, must be evaluated.

To check, if there is a real root in $[t_{\min}, t_{\max}]$, the extrema of the polynomial are first analyzed, as proposed in [74]. The extrema can easily be determined and evaluated using analytical expressions for all quadratic and cubic polynomials. The signs of the extremal values and the function values at $t_{\min}$ and $t_{\max}$ determine whether there exists a real root in $[t_{\min}, t_{\max}]$, and, if this is the case, they allow for refinement of the potential solution set. The smallest root can then be found using standard root finding techniques. A hybrid method, which combines bisection and the Newton-Raphson method [92], was used in this work.

Appendix C

Inequalities

Inequality 1.

$$\cos x \leq 1 - \left(\tfrac{2}{\pi}x\right)^2 \quad \forall x \in \left[-\tfrac{\pi}{2}, \tfrac{\pi}{2}\right] \tag{C.1}$$

Proof. According to the Weierstraß factorization theorem, the cosine function can be written as [101]

$$\cos x = \prod_{n=1}^{\infty} \left(1 - \frac{4x^2}{\pi^2 (2n-1)^2}\right). \tag{C.2}$$

Bearing in mind that all factors are in the range $[0, 1]$ for $x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, the inequality follows directly by neglecting all factors with $n \geq 2$. The first factor is identical to the right-hand side in (C.1). □

Inequality 2.

$$\sin x \leq x \quad \forall x \geq 0 \tag{C.3}$$

Proof. For $x = 0$ both sides are equal. Hence, it is sufficient to show that the function $f(x) := x - \sin x$ is monotonically increasing. This is the case, if the first derivative is always non-negative, $f'(x) = 1 - \cos x \geq 0$, which is obviously satisfied. □

Inequality 3.

$$1 \leq \frac{2\left(\sin\left(\frac{\pi}{2}x\right) - x\right)}{x\left(1 - x^2\right)} \quad \forall x \in]0, 1[\tag{C.4}$$

Proof. The denominator is positive for $x \in]0, 1[$. Thus, the inequality is equivalent to the next statement which is proved in the following for all $x \in [0, 1]$:

$$\frac{x\left(3 - x^2\right)}{2} \leq \sin\left(\frac{\pi}{2}x\right) \tag{C.5}$$

$$\Leftrightarrow \quad \frac{x^2\left(3 - x^2\right)^2}{4} \leq 1 - \left(\cos\left(\frac{\pi}{2}x\right)\right)^2 \tag{C.6}$$

$$\Leftrightarrow \quad \left(\cos\left(\frac{\pi}{2}x\right)\right)^2 \leq \left(1 - x^2\right)^2 \left(1 - \frac{x^2}{4}\right) \tag{C.7}$$

Using the product representation of the cosine function (C.2):

$$\Leftrightarrow \quad \prod_{n=1}^{\infty}\left(1-\frac{x^2}{(2n-1)^2}\right)^2 \leq \left(1-x^2\right)^2\left(1-\frac{x^2}{4}\right) \tag{C.8}$$

$$\Leftrightarrow \quad \prod_{n=2}^{\infty}\left(1-\frac{x^2}{(2n-1)^2}\right)^2 \leq 1-\frac{x^2}{4} \tag{C.9}$$

Since all factors of this product series are in the range $[0,1]$, it is sufficient to show that:

$$\Leftarrow \quad \left(1-\frac{x^2}{9}\right)^2 \cdot \left(1-\frac{x^2}{25}\right)^2 \leq 1-\frac{x^2}{4} \tag{C.10}$$

$$\Leftrightarrow \quad \left(1-x^2\left(\frac{1}{9}+\frac{1}{25}\right)+x^4\frac{1}{9\cdot 25}\right)^2 \leq 1-\frac{x^2}{4} \tag{C.11}$$

Using $x^4 \leq x^2$ for all $x \in [0,1]$:

$$\Leftarrow \quad \left(1-x^2\left(\frac{1}{9}+\frac{1}{25}-\frac{1}{9\cdot 25}\right)\right)^2 \leq 1-\frac{x^2}{4} \tag{C.12}$$

$$\Leftrightarrow \quad \left(1-x^2\frac{11}{75}\right)^2 \leq 1-\frac{x^2}{4} \tag{C.13}$$

$$\Leftrightarrow \quad 1-x^2\frac{22}{75}+x^4\frac{11^2}{75^2} \leq 1-\frac{x^2}{4} \tag{C.14}$$

Using again $x^4 \leq x^2$:

$$\Leftarrow \quad 1-x^2\left(\frac{22}{75}-\frac{11^2}{75^2}\right) \leq 1-\frac{x^2}{4} \tag{C.15}$$

$$\Leftrightarrow \quad 1-x^2\frac{1529}{5625} \leq 1-\frac{x^2}{4} \tag{C.16}$$

$$\Leftarrow \quad \frac{1529}{5625} > \frac{1}{4} \tag{C.17}$$

□

Bibliography

[1] S. Abdollahi-Alibeik, J. P. McVittie, K. C. Saraswat, V. Sukharev, and P. Schoenborn. Analytical modeling of silicon etch process in high density plasma. *Journal of Vacuum Science and Technology A*, 17(5):2485–2491, 1999.

[2] D. Abrahams and A. Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)*. Addison-Wesley Professional, 2004.

[3] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[4] D. Adalsteinsson and J. A. Sethian. A level set approach to a unified model for etching, deposition, and lithography I: Algorithms and two-dimensional simulations. *Journal of Computational Physics*, 120(1):128–144, 1995.

[5] D. Adalsteinsson and J. A. Sethian. A level set approach to a unified model for etching, deposition, and lithography II: Three-dimensional simulations. *Journal of Computational Physics*, 122(2):348–366, 1995.

[6] D. Adalsteinsson and J. A. Sethian. A level set approach to a unified model for etching, deposition, and lithography III: Redeposition, reemission, surface diffusion, and complex simulations. *Journal of Computational Physics*, 138(1):193–223, 1997.

[7] D. Adalsteinsson and J. A. Sethian. The fast construction of extension velocities in level set methods. *Journal of Computational Physics*, 148(1):2–22, 1999.

[8] H. A. Al-Mohssen and N. G. Hadjiconstantinou. Arbitrary-pressure chemical vapor deposition modeling using direct simulation Monte Carlo with nonlinear surface chemistry. *Journal of Computational Physics*, 198(2):617–627, 2004.

[9] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Spring Joint Computer Conference*, pages 483–485, 1967.

[10] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In *An Introduction to Ray Tracing*, pages 201–262. Academic Press, 1989.

[11] S. P. Awate and R. T. Whitaker. An interactive parallel multiprocessor level-set solver with dynamic load balancing. Technical Report UUCS-05-002, School of Computing, University of Utah, 2005.

[12] E. Bär and J. Lorenz. 3-d simulation of LPCVD using segment-based topography discretization. *IEEE Transactions on Semiconductor Manufacturing*, 9(1):67–73, 1996.

[13] F. H. Baumann, D. L. Chopp, T. D. de la Rubia, G. H. Gilmer, J. E. Greene, H. Huang, S. Kodambaka, P. O'Sullivan, and I. Petrov. Multiscale modeling of thin-film deposition: Applications to Si device processing. *MRS Bulletin*, 26(3):182–189, 2001.

[14] R. J. Belen, S. Gomez, D. Cooperberg, M. Kiehlbauch, and E. S. Aydil. Feature-scale model of Si etching in SF_6/O_2 plasma and comparison with experiments. *Journal of Vacuum Science and Technology A*, 23(5):1430–1439, 2005.

[15] R. J. Belen, S. Gomez, M. Kiehlbauch, D. Cooperberg, and E. S. Aydil. Feature-scale model of Si etching in SF_6 plasma and comparison with experiments. *Journal of Vacuum Science and Technology A*, 23(1):99–113, 2005.

[16] M. O. Bloomfield and T. S. Cale. Formation and evolution of grain structures in thin films. *Microelectronic Engineering*, 76(1-4):195–204, 2004.

[17] M. O. Bloomfield, D. F. Richards, and T. S. Cale. A computational framework for modelling grain-structure evolution in three dimensions. *Philosophical Magazine*, 83(31):3549–3568, 2003.

[18] R. Bridson, J. Teran, N. Molino, and R. Fedkiw. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, 21(1):2–18, 2005.

[19] T. S. Cale, M. O. Bloomfield, and M. K. Gobbert. Two deterministic approaches to topography evolution. *Surface and Coatings Technology*, 201(22-23):8873–8877, 2007.

[20] T. S. Cale, T. P. Merchant, L. J. Borucki, and A. H. Labun. Topography simulation for the virtual wafer fab. *Thin Solid Films*, 365(2):152–175, 2000.

[21] T. S. Cale and G. B. Raupp. A unified line-of-sight model of deposition in rectangular trenches. *Journal of Vacuum Science and Technology B*, 8(6):1242–1248, 1990.

[22] S. A. Campbell. *The Science and Engineering of Microelectronic Fabrication*. Oxford University Press, 2nd edition, 2001.

[23] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP*. MIT Press, 2008.

[24] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928.

[25] L. Devroye. *Non-Uniform Random Variate Generation*. Springer, 1986.

[26] F. Dill, A. Neureuther, J. Tuttle, and E. Walker. Modeling projection printing of positive photoresists. *IEEE Transactions on Electron Devices*, 22(7):456–464, 1975.

[27] E. A. Edelberg and E. S. Aydil. Modeling of the sheath and the energy distribution of ions bombarding RF-biased substrates in high density plasma reactors and comparison to experimental measurements. *Journal of Applied Physics*, 86(9):4799–4812, 1999.

[28] B. Engquist and S. Osher. Stable and entropy satisfying approximations for transonic flow calculations. *Mathematics of Computation*, 34(149):45–75, 1980.

[29] M. Fujinaga and N. Kotani. 3-d topography simulator (3-D MULSS) based on a physical description of material topography. *IEEE Transactions on Electron Devices*, 44(2):226–238, 1997.

[30] K. P. Giapis, G. S. Hwang, and O. Joubert. The role of mask charging in profile evolution and gate oxide degradation. *Microelectronic Engineering*, 61-62:835–847, 2002.

[31] J. Glimm, S. R. Simanca, D. Tan, F. M. Tangerman, and G. Vanderwoude. Front tracking simulations of ion deposition and resputtering. *SIAM Journal on Scientific Computing*, 20(5):1905–1920, 1999.

[32] M. K. Gobbert, T. P. Merchant, L. J. Borucki, and T. S. Cale. A multiscale simulator for low pressure chemical vapor deposition. *Journal of the Electrochemical Society*, 144(11):3945–3951, 1997.

[33] M. A. Gosalvez, Y. Xing, K. Sato, and R. M. Nieminen. Atomistic methods for the simulation of evolving surfaces. *Journal of Micromechanics and Microengineering*, 18(5):055029, 2008.

[34] R. A. Gottscho. Ion transport anisotropy in low pressure, high density plasmas. *Journal of Vacuum Science and Technology B*, 11(5):1884–1889, 1993.

[35] R. A. Gottscho, C. W. Jurgensen, and D. J. Vitkavage. Microscopic uniformity in plasma etching. *Journal of Vacuum Science and Technology B*, 10(5):2133–2147, 1992.

[36] B. Gough. *GNU Scientific Library Reference Manual*. Network Theory Ltd., 3rd edition, 2009.

[37] J. Greenwood. The correct and incorrect generation of a cosine distribution of scattered particles for Monte-Carlo modelling of vacuum systems. *Vacuum*, 67(2):217–222, 2002.

[38] S. Hamaguchi, M. Dalvie, R. T. Farouki, and S. Sethuraman. A shock-tracking algorithm for surface evolution under reactive-ion etching. *Journal of Applied Physics*, 74(8):5172–5184, 1993.

[39] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[40] C. Heitzinger. *Simulation and Inverse Modeling of Semiconductor Manufacturing Processes*. Dissertation, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Wien, 2002.

[41] C. Heitzinger, A. Sheikholeslami, F. Badrieh, H. Puchner, and S. Selberherr. Feature-scale process simulation and accurate capacitance extraction for the backend of a 100-nm aluminum/TEOS process. *IEEE Transactions on Electron Devices*, 51(7):1129–1134, 2004.

[42] Y. Hirai, S. Tomida, K. Ikeda, M. Sasago, M. Endo, S. Hayama, and N. Nomura. Three-dimensional resist process simulator PEACE (photo and electron beam lithography analyzing computer engineering system). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(6):802–807, 1991.

[43] A. Hössinger, Z. Djurić, and A. Babayan. Modeling of deep reactive ion etching in a three-dimensional simulation environment. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 53–56, 2007.

[44] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

[45] B. Houston, M. Wiebe, and C. Batty. RLE sparse level sets. In *Proceedings of the ACM SIGGRAPH Conference on Sketches and Applications*, page 137, 2004.

[46] Z.-K. Hsiau, E. Kan, J. McVittie, and R. Dutton. Robust, stable, and accurate boundary movement for physical etching and deposition simulation. *IEEE Transactions on Electron Devices*, 44(9):1375–1385, 1997.

[47] H. Huang, G. H. Gilmer, and T. D. de la Rubia. An atomistic simulator for thin film deposition in three dimensions. *Journal of Applied Physics*, 84(7):3636–3649, 1998.

[48] G. S. Hwang and K. P. Giapis. On the origin of the notching effect during etching in uniform high density plasmas. *Journal of Vacuum Science and Technology B*, 15(1):70–87, 1997.

[49] IEEE Computer Society Standards Committee. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985. IEEE Computer Society Press, 1985.

[50] Y. H. Im, M. Bloomfield, C. Sukam, J. Tichy, T. Cale, and J. Seok. Integrated multiscale multistep process simulation. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 307–310, 2003.

[51] M. M. IslamRaja, C. Chang, J. P. McVittie, M. A. Cappelli, and K. C. Saraswat. Two precursor model for low-pressure chemical vapor deposition of silicon dioxide from tetraethylorthosilicate. *Journal of Vacuum Science and Technology B*, 11(3):720–726, 1993.

[52] R. E. Jewett, P. I. Hagouel, A. R. Neureuther, and T. van Duzer. Line-profile resist development simulation techniques. *Polymer Engineering and Science*, 17(6):381–384, 1977.

[53] M. Jones, J. Baerentzen, and M. Sramek. 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

[54] I. Karafyllidis and A. Thanailakis. Simulation of two-dimensional photoresist etching process in integrated circuit fabrication using cellular automata. *Modelling and Simulation in Materials Science and Engineering*, 3(5):629–642, 1995.

[55] I. V. Katardjiev, G. Carter, M. J. Nobes, S. Berg, and H.-O. Blom. Three-dimensional simulation of surface evolution during growth and erosion. *Journal of Vacuum Science and Technology A*, 12(1):61–68, 1994.

[56] E. Kawamura, V. Vahedi, M. A. Lieberman, and C. K. Birdsall. Ion energy distributions in RF sheaths; review, analysis and simulation. *Plasma Sources Science and Technology*, 8(3):R45–R64, 1999.

[57] H.-B. Kim, G. Hobler, A. Steiger, A. Lugstein, and E. Bertagnolli. Full three-dimensional simulation of focused ion beam micro/nanofabrication. *Nanotechnology*, 18(24):245303, 2007.

[58] H.-B. Kim, G. Hobler, A. Steiger, A. Lugstein, and E. Bertagnolli. Level set approach for the simulation of focused ion beam processing on the micro/nano scale. *Nanotechnology*, 18(26):265307, 2007.

[59] D. Kimpton, M. Baida, V. Zhuk, M. Temkin, and I. Chakarov. Multiple type grid approach for 3d process simulation. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 369–372, 2006.

[60] S. Kistler, E. Bär, J. Lorenz, and H. Ryssel. Three-dimensional simulation of ionized metal plasma vapor deposition. *Microelectronic Engineering*, 76(1-4):100–105, 2004.

[61] R. E. Knop. Algorithm 381: Random vectors uniform in solid angle. *Communications of the ACM*, 13(5):326, 1970.

[62] G. Kokkoris, A. G. Boudouvis, and E. Gogolides. Integrated framework for the flux calculation of neutral species inside trenches and holes during plasma etching. *Journal of Vacuum Science and Technology A*, 24(6):2008–2020, 2006.

[63] G. Kokkoris, A. Tserepi, A. G. Boudouvis, and E. Gogolides. Simulation of SiO_2 and Si feature etching for microelectronics and microelectromechanical systems fabrication: A combined simulator coupling modules of surface etching, local flux calculation, and profile evolution. *Journal of Vacuum Science and Technology A*, 22(4):1896–1902, 2004.

[64] D. Kunder and E. Bär. Comparison of different methods for simulating the effect of specular ion reflection on microtrenching during dry etching of polysilicon. *Microelectronic Engineering*, 85(5-6):992–995, 2008.

[65] O. Kwon, H. Jung, Y. t. Kim, I. Yoon, and T. Won. Level-set modeling of sputter deposition. *Journal of the Korean Physical Society*, 40(1):72–76, 2002.

[66] E. P. Lafortune and Y. D. Willems. Using the modified Phong reflectance model for physically based rendering. Report CW 197, Department of Computing Science, Katholieke Universiteit Leuven, 1994.

[67] F. Lärmer and A. Schilp. Patent numbers DE4241045 (Germany, issued 5 December 1992), US5,501,893 (U.S., issued 26 March 1996).

[68] H. Liao and T. S. Cale. Three-dimensional simulation of an isolation trench refill process. *Thin Solid Films*, 236(1-2):352–358, 1993.

[69] C.-C. Lin and Y.-T. Ching. An efficient volume-rendering algorithm with an analytic approach. *The Visual Computer*, 12(10):515–526, 1996.

[70] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH Computer Graphics*, 21(4):163–169, 1987.

[71] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3):457–462, 2004.

[72] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.

[73] R. Malladi, J. Sethian, and B. Vemuri. Shape modeling with front propagation: A level set approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):158–175, 1995.

[74] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. Fast and accurate ray-voxel intersection techniques for iso-surface ray tracing. In *Proceedings of the International Fall Workshop on Vision, Modeling, and Visualization (VMV)*, pages 429–435, 2004.

[75] G. Marsaglia. Choosing a point from the surface of a sphere. *Annals of Mathematical Statistics*, 43(2):645–646, 1972.

[76] N. Matsunami, Y. Yamamura, Y. Itikawa, N. Itoh, Y. Kazumata, S. Miyagawa, K. Morita, R. Shimizu, and H. Tawara. Energy dependence of the ion-induced sputtering yields of monatomic solids. *Atomic Data and Nuclear Data Tables*, 31(1):1–80, 1984.

[77] S. Mauch. A fast algorithm for computing the closest point and distance transform. Caltech ASCI Technical Report 077, California Institute of Technology, 2000.

[78] G. Mazaleyrat, A. Estve, L. Jeloaica, and M. Djafari-Rouhani. A methodology for the kinetic Monte Carlo simulation of alumina atomic layer deposition onto silicon. *Computational Materials Science*, 33(1-3):74–82, 2005.

[79] S. Mazumder. Methods to accelerate ray tracing in the Monte Carlo method for surface-to-surface radiation transport. *Journal of Heat Transfer*, 128(9):945–952, 2006.

[80] T. P. Merchant, M. K. Gobbert, T. S. Cale, and L. J. Borucki. Multiple scale integrated modeling of deposition processes. *Thin Solid Films*, 365(2):368–375, 2000.

[81] R. Milne. *An adaptive level set method*. PhD thesis, Lawrence Berkeley National Laboratory, University of California, 1995.

[82] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. In *Proceedings of the International Meshing Roundtable*, pages 103–114, 2003.

[83] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[84] M. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, 2006.

[85] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 2003.

[86] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79(1):12–49, 1988.

[87] S. Osher and C.-W. Shu. High-order essentially nonoscillatory schemes for Hamilton–Jacobi equations. *SIAM Journal on Numerical Analysis*, 28(4):907–922, 1991.

[88] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings of the IEEE Visualization Conference (VIS)*, pages 233–238, 1998.

[89] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: Concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.

[90] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[91] J. D. Plummer, M. D. Deal, and P. B. Griffin. *Silicon VLSI Technology*. Prentice Hall Press, 2000.

[92] W. H. Press. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[93] W. Pyka. *Feature Scale Modeling for Etching and Deposition Processes in Semiconductor Manufacturing*. Dissertation, Fakultät für Elektrotechnik, Technische Universität Wien, 2000.

[94] B. Radjenović, S. J. Kim, and J. K. Lee. 3d etching profile evolution simulation using sparse field level set method. In *Proceedings of the International Congress on Plasma Physics (ICPP)*, 2004.

[95] B. Radjenović and J. K. Lee. 3d feature profile simulation for SiO_2 etching in fluorocarbon plasma. In *Proceedings of the International Conference on Phenomena in Ionized Gases (ICPIG)*, 17-142, 2005.

[96] B. Radjenović and M. Radmilović-Radjenović. 3d simulations of the profile evolution during anisotropic wet etching of silicon. *Thin Solid Films*, 517(14):4233–4237, 2009.

[97] B. Radjenović, M. Radmilović-Radjenović, and P. Beličev. 3d simulations with fields and particles. *WSEAS Transactions on Information Science and Applications*, 3(5):869–877, 2006.

[98] B. Radjenović, M. Radmilović-Radjenović, and M. Mitric. Nonconvex Hamiltonians in three dimensional level set simulations of the wet etching of silicon. *Applied Physics Letters*, 89(21):213102, 2006.

[99] S. Reyntjens and R. Puers. A review of focused ion beam applications in microsystem technology. *Journal of Micromechanics and Microengineering*, 11(4):287–300, 2001.

[100] F. Ris, E. Barkmeyer, C. Schaffert, and P. Farkas. When floating-point addition isn't commutative. *ACM SIGNUM Newsletter*, 28(1):8–13, 1993.

[101] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 3rd edition, 1987.

[102] L. Santaló. *Integral Geometry and Geoemtric Probability*. Cambridge University Press, 2nd edition, 2004.

[103] K. Sato, M. Shikida, Y. Matsushima, T. Yamashiro, K. Asaumi, Y. Iriye, and M. Yamamoto. Characterization of orientation-dependent etching properties of single-crystal silicon: effects of KOH concentration. *Sensors and Actuators A: Physical*, 64(1):87–93, 1998.

[104] J. Saussac, J. Margot, and M. Chaker. Profile evolution simulator for sputtering and ion-enhanced chemical etching. *Journal of Vacuum Science and Technology A*, 27(1):130–138, 2009.

[105] E. Scheckler and A. Neureuther. Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):219–230, 1994.

[106] E. Scheckler, N. Tam, A. Pfau, and A. Neureuther. An efficient volume-removal algorithm for practical three-dimensional lithography simulation with experimental verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(9):1345–1356, 1993.

[107] G. F. Schrack. Remark on algorithm 381. *Communications of the ACM*, 15(6):468, 1972.

[108] J. Sethian and D. Adalsteinsson. An overview of level set methods for etching, deposition, and lithography development. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):167–184, 1997.

[109] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Sciences*, pages 1591–1595, 1996.

[110] J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[111] A. Sheikholeslami. *Topography Simulation of Deposition and Etching Processes*. Dissertation, Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Wien, 2006.

[112] T. Shimada, T. Yagisawa, and T. Makabe. Modeling of feature profile evolution in SiO_2 as functions of radial position and bias voltage under competition among charging, deposition, and etching in two-frequency capacitively coupled plasma. *Japanese Journal of Applied Physics*, 45(11):8876–8882, 2006.

[113] T. Shimada, T. Yagisawa, and T. Makabe. Self-consistent modeling of feature profile evolution in plasma etching and deposition. *Japanese Journal of Applied Physics*, 45(5):L132–L134, 2006.

[114] P. Shirley, M. Ashikhmin, M. Gleicher, S. Marschner, E. Reinhard, K. Sung, W. Thompson, and P. Willemsen. *Fundamentals of Computer Graphics*. AK Peters, Ltd., 2nd edition, 2005.

[115] P. Shirley, K. Sung, and W. Brown. A ray tracing framework for global illumination systems. In *Proceedings of the Graphics Interface Conference*, pages 117–128, 1991.

[116] A. Shumilov and I. Amirov. Modeling of deep grooving of silicon in the process of plasmochemical cyclic etching/passivation. *Russian Microelectronics*, 36(4):241–250, 2007.

[117] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Professional, 2002.

[118] T. Smy, S. K. Dew, and R. V. Joshi. Efficient modeling of thin film deposition for low sticking using a three-dimensional microstructural simulator. *Journal of Vacuum Science and Technology A*, 19(1):251–261, 2001.

[119] J. Sreevalsan-Nair, L. Linsen, and B. Hamann. Topologically accurate dual isosurfacing using ray intersection. *Journal of Virtual Reality and Broadcasting*, 4(4), 2007.

[120] C. Steinbruchel. Universal energy dependence of physical and ion-enhanced chemical etch yields at low ion energy. *Applied Physics Letters*, 55(19):1960–1962, 1989.

[121] J. Strain. Tree methods for moving interfaces. *Journal of Computational Physics*, 151(2):616–648, 1999.

[122] E. Strasser. *Simulation von Topographieprozessen in der Halbleiterfertigung*. Dissertation, Fakultät für Elektrotechnik, Technische Universität Wien, 1994.

[123] E. Strasser and S. Selberherr. Algorithms and models for cellular based topography simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(9):1104–1114, 1995.

[124] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 3rd edition, 2000.

[125] G. Sun, X. Zhao, H. Zhang, L. Wang, and G. Lu. 3-d simulation of Bosch process with voxel-based method. In *Proceedings of the IEEE International Conference on Nano/Micro Engineered and Molecular Systems (IEEE-NEMS)*, pages 45–49, 2007.

[126] V. Suvorov, A. Hössinger, Z. Djurić, and N. Ljepojevic. A novel approach to three-dimensional semiconductor process simulation: Application to thermal oxidation. *Journal of Computational Electronics*, 5(4):291–295, 2006.

[127] Synopsys. Sentaurus Topography (Advanced topography simulator). `http://www.synopsys.com/tools/tcad/capsulemodule/sentaurus_topo_ds.pdf`, 2006.

[128] Y. Tan, R. Zhou, H. Zhang, G. Lu, and Z. Li. Modeling and simulation of the lag effect in a deep reactive ion etching process. *Journal of Micromechanics and Microengineering*, 16(12):2570–2575, 2006.

[129] J. Teran, N. Molino, R. Fedkiw, and R. Bridson. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, 21(1):2–18, 2005.

[130] C. Terboven, D. an Mey, and S. Sarholz. OpenMP on multicore architectures. In *A Practical Programming Model for the Multi-Core Era*, pages 54–64. Springer, 2008.

[131] K. Toh, A. Neureuther, and E. Scheckler. Algorithms for simulation of three-dimensional etching. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(5):616–624, 1994.

[132] A. A. Tseng. Recent developments in micromilling using focused ion beam technology. *Journal of Micromechanics and Microengineering*, 14(4):R15–R34, 2004.

[133] I. Utke, P. Hoffmann, and J. Melngailis. Gas-assisted focused electron beam and ion beam processing and fabrication. *Journal of Vacuum Science and Technology B*, 26(4):1197–1276, 2008.

[134] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 61–69, 2006.

[135] R. T. Whitaker. A level-set approach to 3d reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, 1998.

[136] Y. Yamamura, Itakawa, and N. Y., Itoh. Angular dependence of sputtering yields of monatomic solids. Technical Report IPPJ-AM26, Institute of Plasma Physics, Nagoya University, 1983.

[137] R. Zhou, H. Zhang, Y. Hao, and Y. Wang. Simulation of the Bosch process with a string-cell hybrid method. *Journal of Micromechanics and Microengineering*, 14(7):851–858, 2004.

[138] R. Zhou, H. Zhang, Y. Hao, D. Zhang, and Y. Wang. Simulation of profile evolution in etching-polymerization alternation in DRIE of silicon with SF_6/C_4F_8. In *Proceedings of the IEEE International Conference on Micro Electro Mechanical Systems (MEMS)*, pages 161–164, 2003.

[139] Z. F. Zhou, Q. A. Huang, W. H. Li, and W. Lu. A novel 3-d dynamic cellular automata model for photoresist-etching process simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(1):100–114, 2007.

List of Publications

[A20] O. Ertl and S. Selberherr. Three-dimensional level set based Bosch process simulations using ray tracing for flux calculation. *Microelectronic Engineering*, 87(1):20–29, 2010.

[A19] O. Ertl and S. Selberherr. Three-dimensional plasma etching simulation using advanced ray tracing and level set techniques. *ECS Transactions*, 23(1):61–68, 2009.

[A18] O. Ertl and S. Selberherr. A fast level set framework for large three-dimensional topography simulations. *Computer Physics Communications*, 180(8):1242–1250, 2009.

[A17] O. Ertl and S. Selberherr. A fast void detection algorithm for three-dimensional deposition simulation. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 174–177, 2009.

[A16] O. Ertl and S. Selberherr. Three-dimensional topography simulation using advanced level set and ray tracing methods. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 325–328, 2008.

[A15] A. Goncharov, T. Schrefl, G. Hrkac, J. Dean, S. Bance, D. Suess, O. Ertl, F. Dorfbauer, and J. Fidler. Recording simulations on graded media for area densities of up to 1 Tbit/in^2. *Applied Physics Letters*, 91(22):222502, 2007.

[A14] J. Cervenka, H. Ceric, O. Ertl, and S. Selberherr. Three-dimensional sacrificial etching. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 433–436, 2007.

[A13] O. Ertl, C. Heitzinger, and S. Selberherr. Efficient coupling of Monte Carlo and level set methods for topography simulation. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 417–420, 2007.

[A12] T. Schrefl, G. Hrkac, S. Bance, D. Suess, O. Ertl, and J. Fidler. Numerical methods in micromagnetics (finite element method). In H. Kronmueller and S. Parkin, editors, *Handbook of Magnetism and Advanced Magnetic Materials*, volume 2, pages 765–794. Wiley-VCH, 2007.

[A11] F. Dorfbauer, T. Schrefl, M. Kirschner, G. Hrkac, D. Suess, O. Ertl, and J. Fidler. Nanostructure calculation of CoAg core-shell clusters. *Journal of Applied Physics*, 99(8):08G706, 2006.

[A10] O. Ertl, G. Hrkac, D. Suess, M. Kirschner, F. Dorfbauer, J. Fidler, and T. Schrefl. Multiscale micromagnetic simulation of giant magnetoresistance read heads. *Journal of Applied Physics*, 99(8):08S303, 2006.

[A9] J. Fidler, T. Schrefl, D. Suess, O. Ertl, M. Kirschner, and G. Hrkac. Full micromagnetics of recording on patterned media. *Physica B: Condensed Matter*, 372(1-2):312–315, 2006.

[A8] T. Schrefl, D. Suess, G. Hrkac, M. Kirschner, O. Ertl, R. Dittrich, and J. Fidler. Nanomagnetic simulations. In D. Sellmyer and R. Skomski, editors, *Advanced Magnetic Nanostructures*, pages 91–118. Springer, 2006.

[A7] G. Hrkac, M. Kirschner, F. Dorfbauer, D. Suess, O. Ertl, J. Fidler, and T. Schrefl. Three-dimensional micromagnetic finite element simulations including eddy currents. *Journal of Applied Physics*, 97(10):10E311, 2005.

[A6] O. Ertl, T. Schrefl, D. Suess, and M. E. Schabes. Influence of the Gilbert damping constant on the flux rise time of write head fields. *Journal of Magnetism and Magnetic Materials*, 290-291(1):518–521, 2005.

[A5] R. Dittrich, T. Schrefl, M. Kirschner, D. Suess, G. Hrkac, F. Dorfbauer, O. Ertl, and J. Fidler. Thermally induced vortex nucleation in permalloy elements. *IEEE Transactions on Magnetics*, 41(10):3592–3594, 2005.

[A4] G. Hrkac, T. Schrefl, O. Ertl, D. Suess, M. Kirschner, F. Dorfbauer, and J. Fidler. Influence of eddy current on magnetization processes in submicrometer permalloy structures. *IEEE Transactions on Magnetics*, 41(10):3097–3099, 2005.

[A3] M. Schabes, T. Schrefl, D. Suess, and O. Ertl. Dynamic micromagnetic studies of anisotropy effects in perpendicular write heads. *IEEE Transactions on Magnetics*, 41(10):3073–3075, 2005.

[A2] T. Schrefl, M. Schabes, D. Suess, O. Ertl, M. Kirschner, F. Dorfbauer, G. Hrkac, and J. Fidler. Partitioning of the perpendicular write field into head and SUL contributions. *IEEE Transactions on Magnetics*, 41(10):3064–3066, 2005.

[A1] D. Suess, T. Schrefl, M. Kirschner, G. Hrkac, F. Dorfbauer, O. Ertl, and J. Fidler. Optimization of exchange spring perpendicular recording media. *IEEE Transactions on Magnetics*, 41(10):3166–3168, 2005.

Curriculum Vitae

Oct. 2000 – June 2005
Vienna University of Technology, Austria
Study of Technical Physics
Master thesis: "Read Back Signals in Magnetic Recording"
Graduation with excellence

July 2005 – Sept. 2005
Vienna University of Technology, Austria
Research Assistant at the Institute for Solid State Physics

Oct. 2005 – Sept. 2006
Compulsory social service

since Oct. 2006
Vienna University of Technology, Austria
PhD program at the Institute for Microelectronics

May 2008 – July 2008
Internship with Intel in Hillsboro, OR, USA