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

Efficient Flux Calculations 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

Paul Ludwig Manstetten

Matrikelnummer 01528862

geboren am 17. September 1984 in Berlin, Deutschland

Wien, im Juni 2018

Abstract

Within the scope of semiconductor processing, topography simulation is used to predict the interface evolution of a material stack built on top of a wafer. The involved processing steps can be very different, e.g., a new material layer is deposited on the wafer or an existing material stack is etched. Regardless of the process, the continued scaling of microelectronic devices and circuits into the single digit nanometer regime requires high accuracy simulations for complex geometries and process parameters.

Considering a typical dry etching process, the wafer is situated in a vacuum reactor and the full wafer surface is exposed to a low pressure gas phase composed of one or more process gases. Additionally, ions are accelerated vertically towards the wafer and *bombard* the surface in a specific manner. This is typically used to achieve a directional etching on exposed regions of the surface following specific fabrication process parameters.

Predicting such processes on a feature-scale level requires high accuracy three-dimensional simulations which are – as a consequence – computationally very expensive. Therefore, the focus of this work is to accelerate such topography simulations to reduce the simulation runtime, allowing to ultimately increase the pace of research and development of novel devices and circuits.

For high accuracy simulations, the runtime is dominated by the particle transport, which describes, e.g., the amount of etchant particles arriving at a specific region of a surface. Therefore, in this work, computational techniques to accelerate the particle transport for three-dimensional feature-scale process simulations are investigated.

A simulation platform, capable of multi-material advection, is developed and used to implement, and validate the presented approaches. The approaches include

- a reduction of the floating point arithmetics used during the calculation of the ballistic trajectories of the particles,
- an adaptive integration scheme to reduce the number of necessary visibility tests towards the particle sources,
- an iterative partitioning of the integration points on the surface, and
- a one-dimensional approximation of the particle flux for symmetric surface geometries.

Finally, the presented approaches are applied in conjunction to a three-dimensional multi-material etching simulation. It is shown that the overall speedup achieved is above 14 for a wide range of configurations.

Kurzfassung

In der Halbleiterfertigung wird mithilfe von Topographiesimulationen die Veränderung der sich auf dem Wafer befindlichen Materialstrukturen berechnet. Die simulierten Prozessschritte können sehr unterschiedlich sein. Zwei Beispiele sind das Abscheiden einer neuen Materialschicht auf einer bestehenden Struktur oder das Ätzen einer aus mehreren Materialien bestehenden Struktur. Unabhängig vom Prozess sind durch die immer weiter voranschreitende Miniaturisierung der zu fertigenden Strukturen – wobei sich die kleinsten Abmessungen auf den einstellingen Nanometer-Bereich zubewegen – Simulationen mit hoher Genauigkeit für komplexe Geometrien und Prozessparameter nötig.

Bei einem typischen Trockenätz-Verfahren befindet sich der Wafer in einem Vakuumreaktor und die ganze Waferoberfläche wird einer aus den Prozessgasen bestehenden Niederdruck-Gasphase ausgesetzt. Zusätzlich werden Ionen vertikal beschleunigt um die Oberflächenreaktion selektiv zu beinflussen. Solch ein Verfahren wird typischerweise eingesetzt, um ein durch Prozessparameter gesteuertes direktionales Ätzverhalten in exponierten Regionen der Oberfläche zu erreichen.

Um solche Prozesse zu simulieren, werden dreidimensionale Topographiesimulationen auf Strukturebene verwendet, die viel Rechenzeit in Anspruch nehmen. Ziel dieser Arbeit ist es deshalb, solche Topographiesimulationen zu beschleunigen und somit zur Erforschung und Entwicklung von neuartigen Halbleiterbauelementen beizutragen.

Bei Simulationen mit hoher Genauigkeit wird die Laufzeit durch die Berechnung des Partikeltransports, der die Verteilung der am Prozess beteiligten Partikelarten auf der Strukturoberfläche berschreibt, dominiert. In dieser Arbeit werden deshalb Techniken zur Beschleunigung des Partikeltransports in dreidimensionalen Prozesssimulationen auf Strukturebene untersucht.

Eine Simulationsplattform, geeignet für die Simulation von aus mehreren Materialien bestehenden Strukturen, wird entwickelt und eingesetzt, um die vorgestellten Ansätze zu implementieren und zu validieren. Die Ansätze umfassen

- eine Reduktion der arithmetischen Genauigkeit für die Berechnung der ballisitischen Trajektorien der Partikel,
- eine adaptives Integrationsschema, um die Anzahl von notwendigen Sichtbarkeitstests bezüglich der Partikelquellen zu reduzieren,
- eine iterative Partitionierung der Integrationspunkte auf der Strukturoberfläche, und
- eine eindimensionale Approximation des Partikelflusses für Strukturen mit symmetrischer Oberfläche.

Schlussendlich kommen die vorgestellten Ansätze kombiniert in einer dreidimensionalen Simulation eines Ätzprozesses zum Einsatz. Es wird gezeigt, dass für eine große Auswahl an Konfigurationen eine Gesamtbeschleunigung der Simulation um einen Faktor größer 14 erreicht wird.

Acknowledgement

I always enjoyed being a member of the Institute for Microelectronics and I want to thank especially my advisor Siegfried Selberherr (who founded the institute 30 years ago) for providing his full support and an excellent working environment.

I want to thank Josef Weinbub (who leads the Christian Doppler Laboratory for High Performance Technology Computer-Aided Design) and Andreas Hössinger (Silvaco) for a very successful approach to unite the demands in a joint research project in a fruitful manner.

Furthermore, I want to thank the whole staff (and former staff I met) of the institute and especially Lado, Xaver, Luiz, Lukas, Georgios, and Vito.

Contents

Chapter 1

Introduction

Comparing the computational capabilities of today's smartphones to computers from one or two decades ago, the influence of semiconductor technology on our lives is unquestionable. Today, in 2018, a high-end smartphone is built around a hexa-core or octa-core processor[1] running at a clock speed of roughly 2 GHz and equipped with about 8 MB of cache. The processor's transistors are created using semiconductor manufacturing processes with a metal pitch[2] of about 35 nm. This results in a total of a few billion transistors on the processor die interconnected with several kilometers of metal lines. In 2008, similar computing power could be found in a high-end desktop computer with a quad-core processor[3] and 8 MB of cache running at a clock speed of around 3 GHz. This by now outdated processor consisted of about 1 billion transistors with a metal pitch of 160 nm. In 1998, a device with similar computing power would have found its place somewhere in the second half of the famous list of the world's top 500 supercomputers. Back in the day, in this list, the top-ranked supercomputers consisted of hundreds of interconnected single-core processors[4]. The metal pitch was around 500 nm and processors were clocked around 200 MHz.

The production of reliable and energy-efficient structures for today's microprocessors is made possible by the expertise of engineers and researchers at universities and in the semiconductor industry. Most of today's production processes are conducted in meticulously calibrated reactor setups and are sensitive to variations produced by the preceding processing steps. As a consequence, optimizing a production process or developing a new sequence of processes has become a very expensive endeavor. Thus, computer simulations are more and more used to partly replace expensive and lengthy experimental process runs. In all areas of semiconductor product development, computer simulations have become an integral part and are also key to gain additional insights into the utilized processes.

An important branch of simulation-based electronic design automation is technology computer-aided design (TCAD) which models the fabrication and the operation of semiconductor devices and circuits. The modeling of the fabrication is

[1] Apple A11 Bionic (hexa-core), Samsung Exynos 9 Series (octa-core)
[2] The minimum lateral distance between 2 transistor contacts.
[3] Intel i7 CPU 940 (quad-core)
[4] Sun UltraSPARC I/II, IBM POWER2 SC

called *Process TCAD* and includes simulations of etching, deposition, diffusion, and implantation processing steps. The simulated device structures are forwarded to *Device TCAD* simulations to determining the electrical characteristics, which in turn are used by *Circuit TCAD* to simulate the behavior of electronic circuits consisting of multiple interconnected devices.

This work focuses on specific aspects of *Process TCAD*, where nowadays the decreasing physical dimensions of a single device combined with vertical design layouts demand for three-dimensional simulations. For example, if a three-dimensional geometry cannot be approximated as constant in one dimension, the influence of the surrounding geometry on the etch rates (caused by shadowing of parts of the surface) of a plasma etching process (a key topography-changing process) cannot be considered in a two-dimensional simulation. When keeping the resolution on the surface constant, a three-dimensional simulation increases the computational demands. This is noticeable especially when considering the simulation times for etching processes. The accurate calculation of the etch rates on a highly resolved surface is the primary bottleneck, accounting for the majority of the total runtime of the overall simulation. The models for the etch rates depend on the local surface rates of the etchants. In turn, the computational methods used for surface rate calculation constitute nearly the entire computational load of the etch rate calculation [1]. Therein lies a fundamentally important demand for computationally efficient, high performance numerical methods for the surface rate calculation. This demand is the underlying motivation for this work, which focuses on reducing this common computational bottleneck arising in the three-dimensional simulation of an etching process but also in a deposition process.

The remainder of this section lays the groundwork and is arranged as follows: Section 1.1 provides a brief summary of the typical stages in a microprocessor fabrication process. To motivate this research, Section 1.2 introduces an example application of a plasma etching process and its simulation. Finally, the research goals are formulated in Section 1.3 and an outline for the rest of this work is provided in Section 1.4.

1.1 Semiconductor Processing

As this work investigates problems within the general setting of *Process TCAD*, here, typical processing steps are discussed in an abstract manner to provide the context for the discussed topics. The following items provide a brief overview of the processing stages during the production of a microprocessor:

(a) The surface of the silicon wafer is prepared for the first processing steps, e.g., cleaned and polished.

(b) A specific sequence of processing steps is performed to create the transistors at the desired positions. Depending on the technology of the chip generation, the processing sequence is different. Common processing steps include, lithography, etching, deposition, oxidation, implantation, diffusion, and planarization. A majority of the etching and deposition processing steps are conducted in

a vacuum reactor; the goal is to expose the full wafer surface to a controlled environment to maximize the yield, i.e., the portion of properly functioning devices. The gas phase in the reactor is composed out of one or more process gases; the gas composition is chosen to induce the desired surface reaction on the exposed materials stack on the wafer.

(c) By subsequent processing steps, vertical metal contacts to the terminals of the devices (e.g., transistors) are created and the devices are sealed with a dielectric.

(d) The following processing steps create the conducting connections between, for example, the transistor terminals according to the circuit design. To achieve complex connection networks, multiple layers of metal lines, vertically connected through vias, are necessary. The first metal layers start with a pitch size similar to the transistor pitch. The subsequent metal layers have incrementally increasing dimensions, where the dimensions of the outermost metal layer are suitable to connect the processor to the periphery.

(e) The separation of the wafer into single semiconductor chips is the last processing step on the full wafer; it is typically followed by packaging each individual chip into a form suitable for the final application target.

Items (b) to (c) are typically referred to as front-end-of-line; (d) is called back-end-of-line. Both together represent the front-end of the production, while step (e) is referred to as back-end of production.

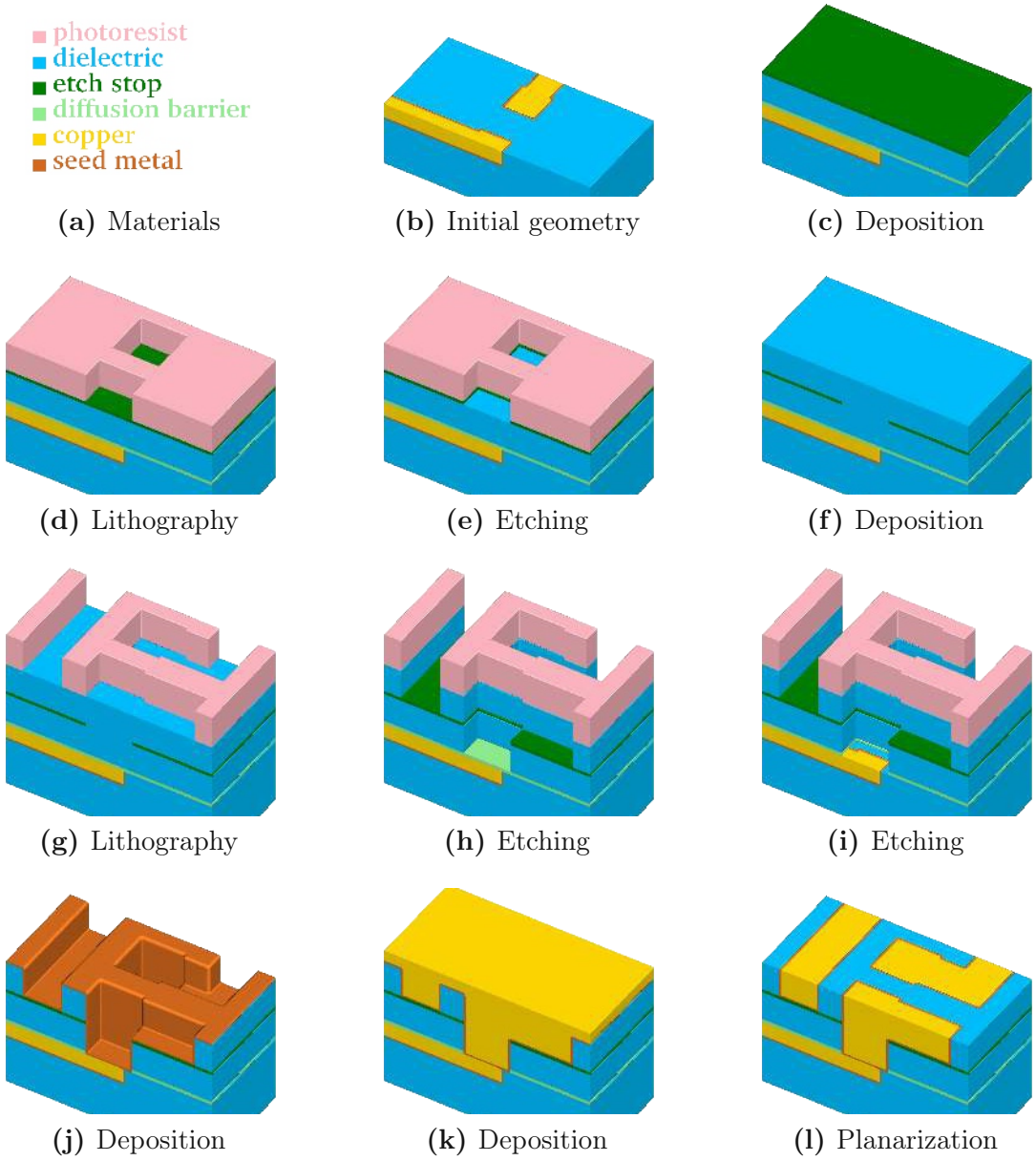
1.2 Motivational Example: Plasma Etching Simulation for Copper Interconnects

Etching is a fundamental processing step. To further outline the challenges and approaches associated with it, an exemplaric etching simulation scenario is introduced and discussed in order to motivate the research presented in this work.

First, a sequence of processing steps, used to fabricate a copper metal layer, is introduced. Then, a brief characterization of a plasma etching process in general is provided by relating it to one of the processing steps, namely the plasma etching of a dielectric layer. It follows a discussion on the demarcation between reactor-scale and feature-scale simulations. Then, an etching simulation of the aforementioned dielectric layer is presented. The commonly involved models and typical choices for computational tasks are briefly discussed and the runtime consumed by these tasks is put in relation.

1.2.1 Copper Interconnect Fabrication

There are various process sequences to fabricate a metal layer above an existing metal layer [2]. All have in common that vertical conducting connections (vias) as well as horizontal connections (lines) have to be created and bonded together.



(a) Materials **(b)** Initial geometry **(c)** Deposition

(d) Lithography **(e)** Etching **(f)** Deposition

(g) Lithography **(h)** Etching **(i)** Etching

(j) Deposition **(k)** Deposition **(l)** Planarization

Figure 1.1: The "self-aligned dual-damascene process" is used to create a copper metal layer. Figures (b) to (l) illustrate all processing steps to create a metalization layer on top of an existing metalization layer (the square domain is clipped along one axis for better visibility). First row (b) to (c): Initial patterned planar conductor and deposition of three layers: A diffusion barrier, a dielectric material, and an etch stop material. Second row (d) to (f): Patterning of the etch stop layer (at the position of the vias) including the deposition of the second dielectric layer. Third row (g) to (i): Patterning of the dielectric layers (vias and lines) including opening of the diffusion barrier layer around the position of the via. Last row (j) to (l): Copper metalization including the deposition of a metal seed layer and chemical-mechanical planarization after the copper deposition.

Depending on the metal used for the connections, the required process sequence can be quite different. Here, we introduce a basic process sequence referred to as "self-aligned dual-damascene process" [2][3]. It is used to process a copper metal layer including vias and lines [4]. The processing steps are visualized in Figure 1.1

for a generic connection layout with three vias; the following describes each step in more detail:

(a) Legend of the different materials involved in the following steps.

(b) Initial surface of the wafer: The lines of the lower metal layer are visible; the position for the vias to the next metal layer are noticeable as the diameter is widened slightly. The copper region is surrounded by a metal seed layer.

(c) Three layers are deposited on the planar surface: A thin barrier to prevent the copper region from diffusing into the dielectric layer, a thick dielectric layer to embed the vias, and a thin etch stop layer on top of the dielectric layer.

(d) A photoresist is patterned on top of the dielectric layer with the positions of the vias.

(e) The pattern of the photoresist is transferred to the etch stop layer.

(f) After the photoresist is removed, a second dielectric layer is deposited to embed the metal lines. The two dielectric layers are now directly connected at the desired positions of the vias.

(g) Again, a photoresist is patterned on top of the dielectric layer, now patterned with the vias and lines.

(h) The pattern of the vias and lines is transferred to the dielectric layers using a single plasma etching process. The images show the result of perfect vertically selective etching and without any etching of the etch stop layer and diffusion barrier.

(i) The exposed area of the diffusion barrier is removed and the underlying copper contact is now open.

(j) A thin layer of seed metal is deposited on the whole surface; it serves as a seed for the crystallization and prevents the copper from diffusing into the dielectric layer.

(k) The cavities for the vias and lines are overfilled with copper using a deposition process.

(l) In a chemical-mechanical-planarization process, the copper above the dielectric layer is removed and a planar wafer surface is obtained; ready to start with the next metalization layer.

1.2.2 Plasma Etching Process

The etching of the dielectric material during the metal layer processing (Figure 1.1h) is in practice realized using a plasma etching process which exposes the wafer surface to a reactive gas phase (plasma) in a vacuum reactor. The gas phase in the reactor is controlled with a gas inlet and a gas outlet leading to a permanent gas stream.

The volatile reaction products are carried away with the gas stream, while other reaction products may stick to the wafer surface or re-deposit on another location. Usually, the etching process is conducted for a predefined time span. A wide variety of plasma etching configurations to achieve various processing goals exist. The configurations differ with regard to the pressure and temperature inside the reactor, the composition of the gas phase, how the plasma is created, and if (or how) the ions in the plasma are accelerated towards the wafer surface.

1.2.3 Simulation Types

Etching process simulations can be partitioned into two major types [5]:

Reactor-Scale. The simulation domain is the full reactor chamber or an important part of it [6], which enables, for example, that uniform properties of the angular and energetic particle distribution in the gas phase over the full surface of the wafer can be an optimization target. Geometric details of the topography on the wafer surface are not modeled explicitly.

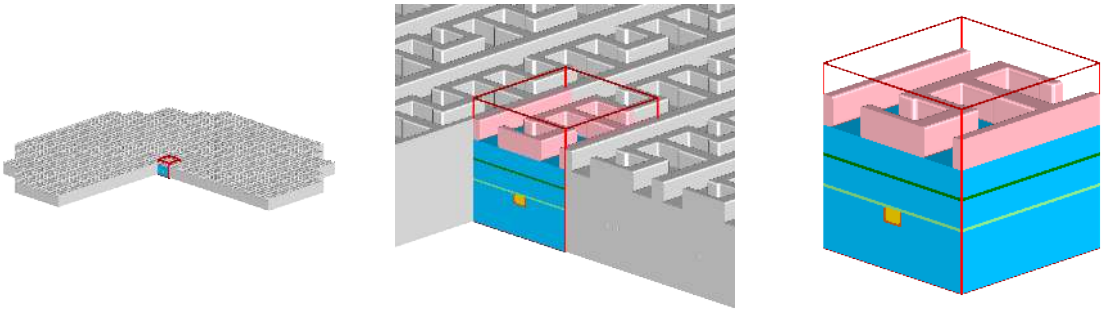
Feature-Scale. The simulation domain is a region of the wafer surface. A common reason for a feature-scale simulation of an etching process is to predict the topographical change of the material stack in the region of interest.

Due to this work's focus on three-dimensional feature-scale simulations, the following provides more details for this type of simulation.

1.2.4 Feature-Scale Etching Simulation

The simulation domain of a feature-scale simulation is usually a rectangular region of the wafer surface containing all geometry details of the stacked materials — forming a device — generated in the preceding processing steps. The lateral boundaries of the simulation domain are usually modeled using reflective or periodic boundary conditions. At the top, towards the reactor chamber, the simulation domain is delimited by a "source plane". By defining angular and energetic distributions of the involved process gas species on the source plane, reactor-scale and feature-scale simulations are decoupled. Figure 1.2 shows an example of a feature-scale simulation domain using the same material stack and geometry introduced in Figure 1.1.

In Figure 1.1h, the result of the etching process is only shown schematically, assuming perfect vertically selective etching and no etching of the other exposed materials. In contrast, Figure 1.3 illustrates the resulting topography when the etching process is simulated in a three-dimensional feature-scale simulation. Without any knowledge about the simulation models in use, it is apparent that all exposed materials are influenced during the etching process. How the topography changes during the etching process is defined by the model for the local etch rates, or more generally, the surface velocities. In advanced models the surface velocity depends on the particle transport of the involved gases inside the feature-scale simulation domain. These advanced models are important for three-dimensional etching and deposition simulations in general and inevitable for high aspect ratio geometries



(a) Larger layout region **(b)** Representative feature **(c)** Simulation domain

Figure 1.2: A larger region of a design layout on a wafer (a), a closer look on a representative feature (b), and the resulting feature-scale simulation domain with indicated domain boundaries in red (c).

(cf. Chapter 7), where the results are sensitive to the accuracy of the solution to the particle transport.

1.2.4.1 Surface Velocity

The particle transport through the gas phase to the surface yields a surface rate distribution for each particle type on the exposed surface. Depending on the particle type, the rate is included in the surface velocity model as a *flux rate* or *sputter rate* for neutral particles and accelerated particles (ions), respectively. In simple models, a linear dependence on the surface rates is assumed for the surface velocity. The normal surface velocity V_n at position $\mathbf{x}$ on the surface can the be written as

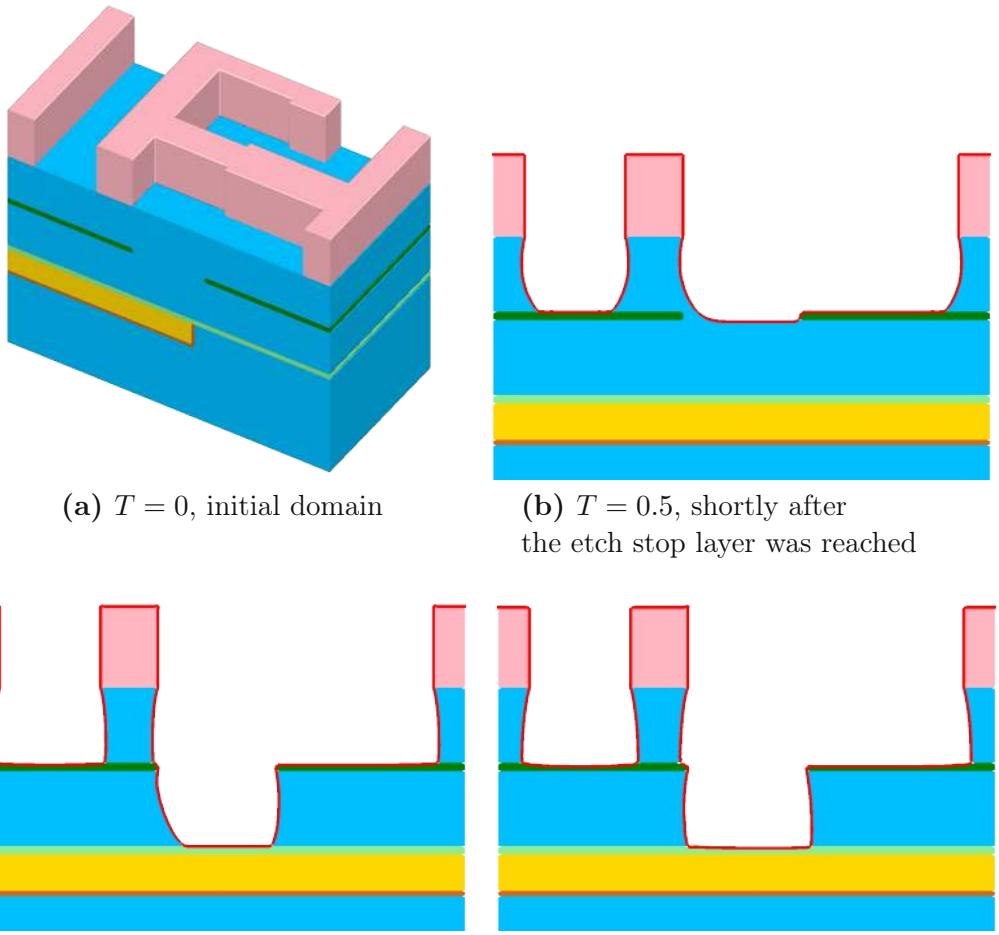
$$V_n(\mathbf{x}) = \sum_{q=1}^{Q} R_q(\mathbf{x}) \cdot C_q(M(\mathbf{x})) \ , \tag{1.1}$$

where Q is the number of simulated particle types, $R_q(\mathbf{x})$ is the surface rate of particle type q at position $\mathbf{x}$, and $C_q(M(\mathbf{x}))$ is a factor depending only on the material M at $\mathbf{x}$.

1.2.4.2 Particle Transport

To obtain the surface rate distributions $R(\mathbf{x})$ on the surface, the transport of the particles from the source plane to the surface must be computed. Due to the low pressures in the reactor chambers and the small geometric extensions of the feature-scale region, particle-particle collisions are typically neglected as the mean free path of a particle is much larger than the domain extensions [5]. This justifies the assumption of ballistic transport of particles and the use of *line-of-sight* methods, if electromagnetic forces on charged particles are neglected.

Depending on the properties of particle sources, material properties, and feature geometry, the incorporation of the re-emission of particles from the surface in the transport model can be very important. For example, in the etching simulation of a



(a) $T = 0$, initial domain

(b) $T = 0.5$, shortly after the etch stop layer was reached

(c) $T = 1.0$, shortly after the diffusion barrier was reached

(d) $T = 1.5$, final topography

Figure 1.3: Results of a feature-scale etching simulation of the dielectric layer in a "self-aligned dual-damascene process" (cf. Figure 1.1h). (a) Initial domain. (b)-(d) Clipped view of the three-dimensional domain for $T = [0.5, 1.0, 1.5]$. The top surface, which is exposed to the process gases, is visualized as red line.

high aspect ratio trench, the vertical surface rate distribution of particles is strongly influenced by the re-emission properties of the sidewalls (cf. Section 7.3).

The numerical methods used to model the particle transport can be divided into three main groups:

(a) In the "straightforward approach", the surface, as well as the source plane, is discretized into N_s elements (e.g., using triangles). Furthermore, also the angular and energetic flux distribution is discretized by an appropriate set of N_f basis functions (e.g., subdivision of the solid angle and a binning scheme for energies). From this discretization, a linear system of equations with $N_s \cdot N_f$ unknowns and $\mathcal{O}(N_s^2)$ non-zero entries is assembled. Constructing the system matrix requires visibility information between all elements.

(b) In the *bottom-up* iterative scheme, the surface is discretized and the direct flux for each surface element is calculated by numerical integration over the solid

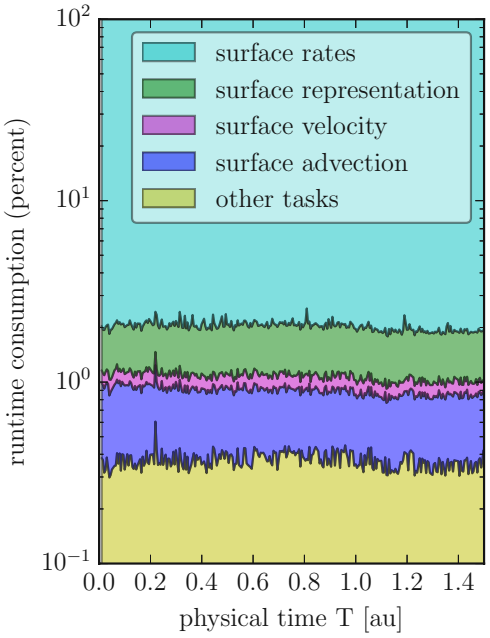


Figure 1.4: Runtime of the main computational tasks (logarithmic scale) for all time steps ($T = 0$ to $T = 1.5$) (cf. Figure 1.3).

angle under which the source plane is visible. For each surface element, an averaged representation of the incoming particles is maintained. A number of iterations is then performed, where the flux rates are updated by numerical integration over the solid angle under which the surface is visible. The number of iterations is equal to the number of considered re-emissions.

(c) In the *top-down* approach, the surface is discretized and particles are traced from random locations on the source plane according to the angular and energetic distribution of the particle sources. At the first intersection with the surface, a contribution to the local surface rate is calculated and a new particle is re-emitted according to the local surface re-emission properties. After a fixed number of re-emissions or a threshold condition for the energy of the particle the tracing is terminated.

All three approaches have in common that a large number of visibility/intersection tests are involved, either providing the information if two points in the domain are mutually visible (not obscured) or which is the first element intersecting a path starting from an element in a certain direction.

The particle transport calculation has to be repeated in each time step of a simulation due to the changed topography of the surface. When using any of the numerical methods listed above in a three-dimensional etching simulation, it constitutes the major part of the computational load in each time step.

1.2.4.3 Surface Advection

The advection of the exposed surfaces is preferentially realized by the utilization of the level-set method (cf. Section 2.1). In contrast to cell-based or mesh-based methods, it provides a robust framework to deal with topological changes of the surface, for example, when a surface splits in two parts or merges with itself. The level-set equation (1.2) relates the temporal change of a scalar level-set function ϕ to the norm of the gradient of ϕ using a scalar velocity field $V_{ext}(\mathbf{x})$ as a prefactor. The velocity field V_{ext} is not only defined on the surface, but in the whole domain. It is constructed via an appropriate extension of the surface velocities V_n; a simple construction scheme is to use the value at the closest point on the surface.

$$\frac{\partial \phi(\mathbf{x})}{\partial t} = V_{ext}(\mathbf{x}) \parallel \nabla \phi(\mathbf{x}) \parallel \tag{1.2}$$

Common practice is to initialize ϕ so that the surface in question is represented implicitly by the zero-level-set of ϕ. The evolution of the surface over time is obtained by solving (1.2). The spatial discretization of $\nabla\phi$ is usually accomplished using suitable higher-order finite-difference schemes on a Cartesian grid [7]. A simple choice for the temporal discretization is the explicit Euler method. To obtain a stable scheme, a maximum time step of

$$\Delta t = \frac{\Delta h}{|V_{ext_{max}}|} \cdot \alpha_{CFL} \tag{1.3}$$

is necessary, where Δh is the spatial grid spacing, $V_{ext_{max}}$ is the maximum value in the velocity field, and $\alpha_{CFL} \in \,]0, 1]$ is a positive constant depending on the discretization (cf. Section 2.3.4.3). As a consequence, for a given physical simulation time, the number of necessary time steps grows linear with the resolution of the surface.

1.2.4.4 Computational Demands

Assuming the level-set method is used for surface advection, the common computational tasks in one time step of a simulation are

(a) the preparation of a suitable surface representation for the surface rate calculations,

(b) the calculation of the surface rate distributions on the surface,

(c) the evaluation of the surface velocity models using the surface rate distributions, and

(d) the advection of the level-set function according to the computed surface velocity field (e.g., solving (1.2)).

Figure 1.4 depicts the runtime for each of these tasks based on an exemplary etching simulation of the dielectric layer shown in Figure 1.3. The simulation is performed on a 16-core workstation (WS2, cf. Section 3.4) and the surface rate calculation is

parallelized (OpenMP, 32 threads) as well as the surface advection. The horizontal resolution of the domain is 192×192 grid cells. The surface velocity is modeled using a vertically focused source of a single particle species according to

$$V_n(\mathbf{x}) = R(\mathbf{x}) \cdot C(M(\mathbf{x})) \ . \tag{1.4}$$

In the particle transport model only the direct flux rate is considered; a *bottom-up* approach with 5120 sampling directions to integrate the arriving particle distribution at each surface element is used. The calculation of the surface rates consumes the major part of the overall runtime of the simulation.

1.3 Research Goals

The goal of this work is to provide techniques to reduce the calculation time of particle transport models – constituting the major computational bottleneck – in three-dimensional feature-scale process simulations. In particular, the focus is on etching and deposition processes. Figure 1.4 shows that even for low surface resolutions and simple models for the particle transport, e.g., only considering the direct flux rate and ignoring contributions from re-emissions, the surface rate calculation by far takes up the majority of the overall runtime of the simulation: A perceptible speedup of the underlying numerical methods of the surface rate calculation would therefore equate to a nearly identical speedup for the overall runtime; this holds for a wide range of etching as well as deposition simulations. Such a runtime reduction is highly desirable as it paves the way for high surface resolutions and the option to integrate advanced models capturing effects neglected before because of unfeasible simulation runtimes.

Another important goal of this work is to develop numerically robust methods for arbitrary geometries, which have a clear interface and can be straightforwardly integrated into existing simulation tools.

Research Setting. The research presented in this work was conducted within the scope of the Christian Doppler Laboratory for High Performance Technology Computer-Aided Design (TCAD). The Christian Doppler Association funds cooperations between companies and research institutions pursuing application-orientated basic research. In this case, the cooperation was established between the Institute for Microelectronics at the TU Wien and Silvaco Inc., a company developing and providing electronic device automation and TCAD software tools.

1.4 Outline of the Thesis

Chapter 2 provides an overview of currently available software frameworks for feature-scale process simulations and introduces commonly used numerical methods for the particle transport, which are a focus for this work. The level-set method (including its common discretization schemes) is introduced in more detail as it is used as foundation for the surface advection in many frameworks.

Chapter 3 introduces the developed simulation platform which is used in all subsequent chapters to implement, validate, and test the accuracy and performance of new numerical approaches for the flux rate calculation.

Chapter 4 presents an approach to accelerate the direct flux calculations in three-dimensional process simulations: Using a subdivided icosahedron to define the search directions, it is shown that the surface rate calculation can be accelerated by a factor of at least two when adaptively refining the search directions only near the aperture boundary. If the refinement is initialized on a level appropriate to the expected topography, the accuracy of the resulting surface rates is not reduced.

Chapter 5 presents an approach to perform the visibility calculations for the surface rate calculation on a temporarily generated explicit surface mesh utilizing modern hardware-tailored single-precision ray tracing frameworks. When compared to ray tracing directly on the level-set data structure, a speedup of at least a factor of 3 is attested.

Chapter 6 presents an approach to evaluate the surface rates only on a sparse set of points on the surface to reduce the computational effort. This sparse set of points is generated according to application-specific requirements using an iterative partitioning scheme. The resulting sparse surface rates are "diffused" in the neighborhood to approximate a linear interpolation on the whole surface. The obtained speedup factors range from 2 to 8 and the produced deviations of the surface position are below 3 grid cells for all tested cases.

Chapter 7 introduces a radiosity-based method to approximate the flux rates of neutral particles in very high aspect ratio holes and trenches. It is based on a one-dimensional approximation of the three-dimensional surface and is eligible if simulation runtime is critical and the geometry can be approximated with a convex rotationally symmetric hole or convex symmetric trenches.

Chapter 8 demonstrates the overall obtained speedup when applying the approaches presented in Chapters 4 to 6 in conjunction.

Finally, Chapter 9 concludes with a brief overall summary and presents ideas for future work.

Chapter 2

Review of Methods for Particle Transport and Surface Evolution

This chapter starts with an overview of some currently available three-dimensional topography/process simulators (Section 2.1). The underlying numerical methods used in these simulators are described, although the level of detail of the available information is limited for closed-source simulators. Nevertheless, recurrent components can be identified as a numerical method to solve the particle transport inside the domain and a numerical scheme to advect the surface. In Section 2.2, three numerical methods to solve the particle transport with the aim to obtain the surface rates (which enter the surface velocity model) are discussed. Section 2.3 introduces the level-set method which is predominantly used for the surface advection. Finally, Section 2.4 concludes this chapter with a brief summary.

2.1 Software and Methods

Three-dimensional topography simulation for etching and deposition processes is available via commercial, open-source, and in-house simulation frameworks. For a selection of publicly known frameworks, a brief description of the relevant methods and numerical approaches is given in the following. For closed-source simulators the brevity of the descriptions stems from the fact that little or no details of their numerical approaches are available.

Victory Process [8] is a commercial general purpose process simulator distributed by *Silvaco*. It features a *Process Simulator Mode* to simulate three-dimensional etching and deposition processes based on physical models. A set of etching and deposition models is provided and user-defined models, which have access to a set of simulated local surface conditions, are supported. The particle transport simulation engine is parallelized via a shared-memory approach. To evolve the surface it is discretized using a Cartesian hierarchical level-set-based representation. In [9] the authors use *Victory Process* to model the Bosch process in a microelectromechanical systems (MEMS) application.

Sentaurus Topography 3D [10] is a commercial profile simulator distributed by *Synopsis*. It features three-dimensional topography simulations and is focused on etching and deposition processes. It has a concept of user-defined models similar to *Victory Process* and also provides a set of predefined models for etching and deposition. The particle transport is modeled with a Monte Carlo approach. The surface representation used for the evolution is level-set-based. Parallelization on shared-memory systems is supported. In [11] the authors use *Sentaurus Topography 3D* to investigate the influence of the photoresist profile on the etching process.

SEmulator3D [12] is a commercial three-dimensional process modeling platform distributed by *Coventor*. It uses an unspecified proprietary evolution engine to evolve the material surfaces. Similar to the above mentioned simulators, it offers a set of predefined models for etching and deposition. In [13] the authors use *SEmulator3D* to assess different approaches for patterning metal lines and vias.

ViennaTS [14][15] is an open-source topography simulator developed and maintained by the *Institute for Microelectronics, TU Wien*. Two- and three-dimensional simulations are supported. The surface is represented as a single resolution level-set discretized on a Cartesian grid. The implementation of the level-set is based on the sparse field level-set method using a hierarchical run-length encoded data structure. The particle transport is implemented using a Monte Carlo approach and an explicit representation of the surface (partially overlapping disks). A set of advanced physical etching and deposition models are implemented; user-defined models require a recompilation. The surface evolution and the particle transport is parallelized for shared-memory systems using OpenMP. In [1] the authors simulate the Bosch process to demonstrate the capabilities of *ViennaTS*. Prior to *ViennaTS*, *Topo3D*[16], also a three-dimensional level-set-based topography simulator, was developed at the *Institute for Microelectronics, TU Wien*.

A selection of recent publications which apply three-dimensional process simulations using closed, in-house frameworks are mentioned below; depending on the publication some details are provided about the numerical approaches taken: In [17] a wet etching process is simulated based on the sparse field level-set method implementation of the *ITK*[18] library extended for arbitrary surface speed functions. The authors of [19] use a narrow band level-set method for the surface evolution and a Monte Carlo method to model the particle transport in a three-dimensional simulation of a deep reactive ion etching (DRIE) process; the surface is represented with small overlapping spheres during the computation of the Monte Carlo particle transport. In [20] a cell-based method seems to be used for the surface evolution (although not explicitly stated) and a Monte Carlo method is used to model the particle transport.

Most simulation frameworks described above provide a set of common parameterized models for the surface velocity but also allow for user-defined model implementations. The underlying reason for the simulation stipulates which models are

appropriate, e.g., etching simulations of high aspect ratio structures rely on accurate models for re-emission and re-deposition of particles. Equation (2.1) is an example for a general formulation for the surface velocity which relates the normal surface velocity V_n to

(a) the volumetric properties of the bulk material M
(e.g., to model the influence of stress types in the bulk material),

(b) the normal direction $\boldsymbol{n}$
(e.g., to model the influence of a crystallographic direction),

(c) the curvature κ
(e.g., to model a smoothing of the surface),

(d) and the surface rate distributions of ions R_{ion} and neutral particles R_{neu}
(e.g., to model the etch rate in an ion-enhanced plasma etching process).

$$V_n(\mathbf{x}) = f(M(\mathbf{x}), \boldsymbol{n}(\mathbf{x}), \kappa(\mathbf{x}), R_{ion}(\mathbf{x}), R_{neu}(\mathbf{x})) \tag{2.1}$$

The surface rate of neutral particles R_{neu} is typically a scalar flux value describing how many particles are locally adsorbed per unit time. For accelerated ions the surface rate R_{ion} typically does not represent a particle flux directly but the effective local sputter rate, e.g., how much of the local material is sputtered away. Models for the sputter rate typically depend on the energy and incoming angle (relative to the surface normal) of the ions.

Often only a subset of dependencies is important to be modeled. If an important dependence for the simulation is identified a subsequent question is how the relevant parameters are obtained. Typically a compromise between simulation runtime and accuracy of the models has to be made. In any of the frameworks described above the particle transport is an integral part of the advanced models for dry etching or deposition processes. The next section concentrates on the numerical methods to approximate the surface rate distributions based on the particle transport.

2.2 Feature-Scale Particle Transport

This section reviews approaches to model the particle transport towards the surface in a feature-scale simulation, eventually resulting in the surface rate distributions which enter the surface velocity model as a parameter.

Any advanced surface velocity model for dry etching or deposition processes depends on how the relevant particle species are distributed. In general, the rate of the particles on the surface (number of particles per time and per area) and their kinetic energy is important. If accelerated ions are modeled, the effect on the surface can depend significantly on the kinetic energy of the ions. For neutral particles (which are not accelerated by an electric field) the energy is less important as the energy levels at usual process temperatures are too small to have a noticeable effect on the surface. The influence of the neutral particles on the surface is mainly

due to the induced chemical reactions which, in turn, are modeled using the rate of the particles.

A general formulation relating the arrival particle flux distribution Γ_{in} to the particle flux distribution which is emitted Γ_{out} using a re-emission function F_{re} is given in (2.2). Additionally, a source term Γ_{src} independent of the arriving particles is considered. N_Q is the number of particle types and Γ_{re} is the re-emitted particle flux distribution. The equation is defined for a point $\boldsymbol{x}$ on the surface at time t, although not explicitly shown for the sake of brevity.

$$\begin{aligned}\Gamma_{out}(\boldsymbol{\omega}_0, Q_0, E_0) &= \Gamma_{src}(\boldsymbol{\omega}_0, Q_0, E_0) + \\ &+ \sum_{q=0}^{N_Q} \underbrace{\left[\int_\Omega \int_E \left[F_{re}((\boldsymbol{\omega}_0, Q_0, E_0), (\boldsymbol{\omega}_{d\Omega}, Q_q, E)) \cdot \Gamma_{in}(\boldsymbol{\omega}_{d\Omega}, Q_q, E) \right] dE d\Omega \right]}_{\Gamma_{re}(\boldsymbol{\omega}_0, Q_0, E_0)}\end{aligned} \tag{2.2}$$

Equation (2.2) states that the flux of particles of type Q_0 and energy E_0 leaving into direction $\boldsymbol{\omega}_0$ is equal to all arriving particles (integrated over all incoming directions Ω and all energy levels E) weighted with a re-emission function F_{re}. Note that F_{re} potentially also re-emitts particles different in energy and type from the incoming particle (Q_q, E), e.g., if accelerated ions bombard the surface and sputter other types of particles situated on the surface.

In the field of computer graphics the *rendering equation* (2.3) [21] takes a form similar to (2.2) for a point $\boldsymbol{x}$ on the surface at time t.

$$\begin{aligned}L_{out}(\boldsymbol{\omega}_0, \lambda) &= L_{src}(\boldsymbol{\omega}_0, \lambda) + \\ &+ \int_\Omega F_{BRDF}(\boldsymbol{\omega}_0, \boldsymbol{\omega}_{d\Omega}, \lambda) \cdot L_{in}(\boldsymbol{\omega}_{d\Omega}, \lambda)(\boldsymbol{\omega}_{d\Omega} \cdot \boldsymbol{n}) d\Omega\end{aligned} \tag{2.3}$$

Equation (2.3) states (for wavelength λ) that the radiance L_{out} emitted into direction $\boldsymbol{\omega}_0$ is equal to the integral of the received radiance L_{in} normalized by the projected area $(\boldsymbol{\omega}_{d\Omega} \cdot \boldsymbol{n})$ and scaled with a bidirectional reflectance distribution function F_{BRDF}. The radiance is the radiant energy per unit time per unit solid angle per unit projected area.

The following assumptions are implicitly made if (2.2) is used as a general starting point to model the particle transport:

(a) The re-emission of particles is not influenced by the arriving particles, e.g., F_{re} is independent of Γ_{in}.

(b) The re-emission occurs instantaneous and at the location where the particles hit the surface, e.g., no transport on/below the surface or time delay is considered.

When taking the perspective of a point on the surface $\mathcal{S}$, Figure 2.1 illustrates the directions for which other parts of the surface and the source plane $\mathcal{P}$ contribute to the arriving flux distribution Γ_{in}, if

(c) particle-particle collisions (which would change the momentum of the involved particles) are neglected due to the low pressure and small dimension of the simulation domain, and

(d) charged particles (ions) are not influenced by the electromagnetic field in the domain which allows to model the trajectory of particles as straight lines.

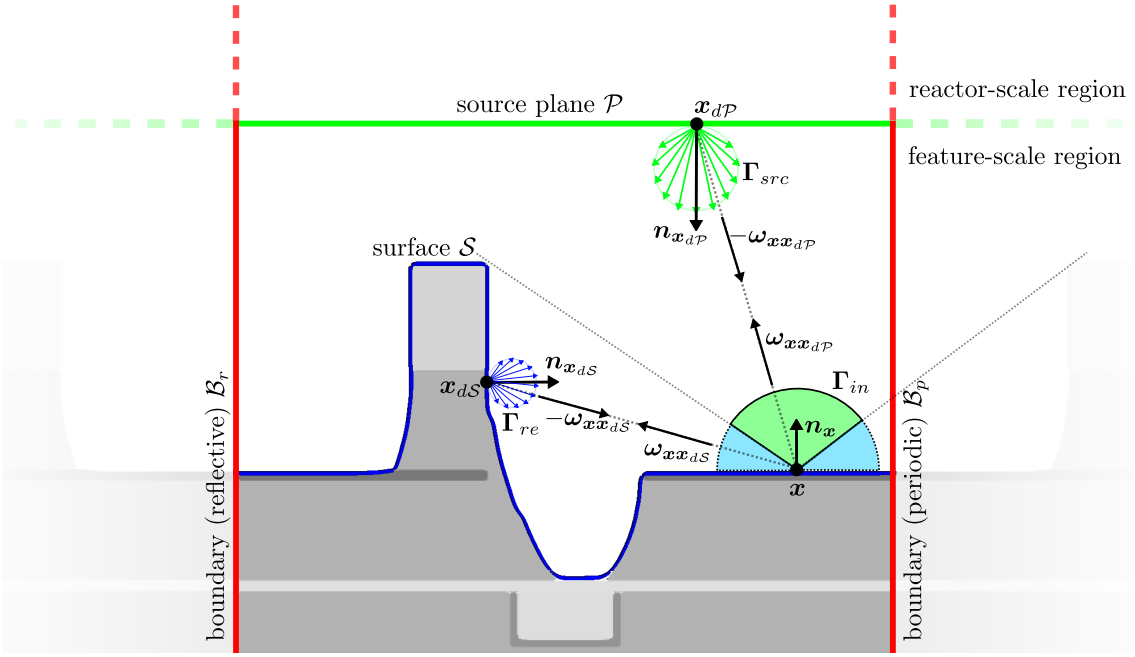


Figure 2.1: The feature-scale simulation domain is delimited by the source plane (green) in upward direction. The lateral boundaries (red) are typically either reflective (left side) or periodic (right side). The surface of the geometry $\mathcal{S}$ is shown in blue. The different shades of gray below the surface indicate different material regions. The arriving flux Γ_{in} is illustrated for point $\boldsymbol{x}$ on $\mathcal{S}$. The arriving flux Γ_{in} is divided into a direct contribution from the source (green segment) and a contribution from re-emitted particles (blue segment). An isotropic (diffuse) emission distribution Γ_{src} is illustrated for a point $\boldsymbol{x}_{d\mathcal{P}}$ on $\mathcal{P}$ and an isotropic re-emission distribution Γ_{re} is illustrated for a point $\boldsymbol{x}_{d\mathcal{S}}$ on $\mathcal{S}$. The surface normal directions $\boldsymbol{n}$ and the direction vectors $\boldsymbol{\omega}$ between the points are shown.

2.2.1 Surface Rate Calculation

The general formulation of the re-emission function F_{re} in (2.2) allows to model numerous effects, e.g., specular-like reflections and sputtering of particles from the surface which themselves are transported in the domain. In [5] a broad overview of common multi-particle models for the surface velocity is provided. In the following only the case of

(a) a single source of a mono-energetic particle species,

(b) a directionally independent probability for adsorption $s(\boldsymbol{x})$,

(c) an isotropic re-emission with probability $1 - s(\boldsymbol{x})$,

(d) and a single scalar surface rate R representing the total arriving particle flux

is considered. Hence, (2.2) becomes

$$\begin{aligned}\Gamma_{out}(\boldsymbol{x},\boldsymbol{\omega}_0) &= \Gamma_{src}(\boldsymbol{x},\boldsymbol{\omega}_0) + \int_\Omega \left[(1-s)\frac{1}{\pi}(\boldsymbol{n_x}\cdot\boldsymbol{\omega}_0)\Gamma_{in}(\boldsymbol{x},\boldsymbol{\omega}_{d\Omega})\right] d\Omega \\ &= \Gamma_{src}(\boldsymbol{x},\boldsymbol{\omega}_0) + \underbrace{(1-s)\frac{1}{\pi}(\boldsymbol{n_x}\cdot\boldsymbol{\omega}_0)\int_\Omega \Gamma_{in}(\boldsymbol{x},\boldsymbol{\omega}_{d\Omega})d\Omega}_{\Gamma_{re}(\boldsymbol{x},\boldsymbol{\omega}_0)}\end{aligned} \tag{2.4}$$

or reformulated when using an integral over the visible source and surface areas

$$\begin{aligned}\int_\Omega \Gamma_{in}(\boldsymbol{x},\boldsymbol{\omega}_{d\Omega})d\Omega = &\int_{\mathcal{P}_{vis}} \frac{\boldsymbol{\omega_{x x_{d\mathcal{P}}}}\cdot\boldsymbol{n_x}}{\parallel \boldsymbol{x}-\boldsymbol{x}_{d\mathcal{P}} \parallel^2}\left[\Gamma_{src}(\boldsymbol{x_{d\mathcal{P}}},-\boldsymbol{\omega_{x x_{d\mathcal{P}}}})\right] d\mathcal{P} + \\ &+\int_{\mathcal{S}_{vis}} \frac{\boldsymbol{\omega_{x x_{d\mathcal{S}}}}\cdot\boldsymbol{n_x}}{\parallel \boldsymbol{x}-\boldsymbol{x}_{d\mathcal{S}} \parallel^2}\left[\Gamma_{re}(\boldsymbol{x_{d\mathcal{S}}},-\boldsymbol{\omega_{x x_{d\mathcal{S}}}}\right] d\mathcal{S}\ .\end{aligned} \tag{2.5}$$

The objective is to solve (2.4) and to obtain the surface rate distributions. The surface rates R_i are calculated by an integration of the arriving flux distribution Γ_{in} weighted by a corresponding weight function r_i.

$$R_i(\boldsymbol{x}) = \int_\Omega r_i(\boldsymbol{\omega}_{d\Omega})\cdot\Gamma_{in}(\boldsymbol{x},\boldsymbol{\omega}_{d\Omega})d\Omega \tag{2.6}$$

In the simplest case (which is the one considered here) only a single rate R is used and $r(\boldsymbol{\omega}_{d\Omega}) = 1$, which reduces (2.6) to (2.7) and corresponds to the total arriving particle flux.

$$R(\boldsymbol{x}) = \int_\Omega \Gamma_{in}(\boldsymbol{x},\boldsymbol{\omega}_{d\Omega})d\Omega \tag{2.7}$$

A simple corresponding surface velocity model is

$$V_n(\mathbf{x}) = f(M(\mathbf{x}),R(\mathbf{x})) = \alpha\cdot R(\boldsymbol{x})\cdot s(\boldsymbol{x})\ , \tag{2.8}$$

where α is a scalar value. In the following three commonly used but conceptually different methods are described which aim to obtain the surface rates R by solving (2.4) or to approximate the solution.

2.2.1.1 Straightforward Approach

The "straightforward approach" solves the particle transport equation beforehand and subsequently extracts the surface rates from the arriving particle flux distributions. The surface $\mathcal{S}$ and the source plane $\mathcal{P}$ are discretized into $N_\mathcal{S}$ and $N_\mathcal{P}$ elements

respectively, e.g., triangles (three dimensions) or lines (two dimensions). The flux distributions Γ are approximated by the superposition of a set of basis functions with corresponding coefficients. In the general case Γ can then be approximated as

$$\Gamma(\boldsymbol{x}, \boldsymbol{\omega}) \approx \sum_i C_i(\boldsymbol{x}) \cdot c_i(\boldsymbol{\omega}) \ , \tag{2.9}$$

where C_i are the scalar coefficients and c_i are the basis functions. If the basis functions are orthonormal the coefficients C_i are defined as

$$C_i(\boldsymbol{x}) = \int_\Omega c_i(\boldsymbol{\omega}_{d\Omega}) \Gamma(\boldsymbol{x}, \boldsymbol{\omega}_{d\Omega}) d\Omega \ . \tag{2.10}$$

For the simplest case, a single constant basis function is used and (2.10) reduces to

$$C(\boldsymbol{x}) = \int_\Omega \Gamma(\boldsymbol{x}, \boldsymbol{\omega}_{d\Omega}) d\Omega = R(\boldsymbol{x}). \tag{2.11}$$

Inserting (2.11) into (2.5) and expressing Γ_{re} according to (2.4) yields

$$\begin{aligned} C_{in}(\boldsymbol{x}) = &\int_{\mathcal{P}_{vis}} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{P}}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{d\mathcal{P}} \|^2} \left[\Gamma_{src}(\boldsymbol{x}_{\boldsymbol{d\mathcal{P}}}, -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{P}}})\right] d\mathcal{P} + \\ &+ \int_{\mathcal{S}_{vis}} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{S}}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{d\mathcal{S}} \|^2} \left[(1-s)\frac{1}{\pi}(\boldsymbol{n}_{\boldsymbol{x}_{d\mathcal{S}}} \cdot -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{S}}}) \cdot C_{in}(\boldsymbol{x}_{\boldsymbol{d\mathcal{S}}})\right] d\mathcal{S} \ . \end{aligned} \tag{2.12}$$

The discretization of $\mathcal{S}$ and $\mathcal{P}$ into elements with area A results in

$$\begin{aligned} C_{in}(\boldsymbol{x}) = &\sum_j \int_{A_j} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{dA_j}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{dA_j} \|^2} \left[\Gamma_{src}(\boldsymbol{x}_{dA_j}, -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{dA_j}})\right] dA_j + \\ &+ \sum_j \int_{A_j} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{dA_j}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{dA_j} \|^2} \left[(1-s)\frac{1}{\pi}(\boldsymbol{n}_{\boldsymbol{x}_{dA_j}} \cdot -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{dA_j}}) \cdot C_{in}(\boldsymbol{x}_{\boldsymbol{dA_j}})\right] dA_j \ , \end{aligned} \tag{2.13}$$

and an integration of the arriving total flux C_{in} for a surface element leads to

$$\begin{aligned} \int_{A_i} C_{in}(\boldsymbol{x}_{dA_i}) dA_i = &\sum_j^{N_\mathcal{P}} \int_{A_i} \int_{A_j} \frac{\boldsymbol{\omega}_{\boldsymbol{x}_{dA_i}\boldsymbol{x}_{dA_j}} \cdot \boldsymbol{n}_{\boldsymbol{x}_{dA_i}}}{\| \boldsymbol{x}_{dA_i} - \boldsymbol{x}_{dA_j} \|^2} \left[\Gamma_{src}(\boldsymbol{x}_{dA_j}, -\boldsymbol{\omega}_{\boldsymbol{x}_{dA_j}\boldsymbol{x}_{dA_j}})\right] dA_j dA_i + \\ &+ \sum_j^{N_\mathcal{S}} \int_{A_i} \int_{A_j} \frac{\boldsymbol{\omega}_{\boldsymbol{x}_{dA_i}\boldsymbol{x}_{dA_j}} \cdot \boldsymbol{n}_{\boldsymbol{x}_{dA_i}}}{\| \boldsymbol{x}_{dA_i} - \boldsymbol{x}_{dA_j} \|^2} \left[(1-s)\frac{1}{\pi}(\boldsymbol{n}_{\boldsymbol{x}_{dA_j}} \cdot -\boldsymbol{\omega}_{\boldsymbol{x}_{dA_j}\boldsymbol{x}_{dA_j}}) \cdot C_{in}(\boldsymbol{x}_{dA_j})\right] dA_j dA_i \ . \end{aligned} \tag{2.14}$$

In this case only isotropic (diffuse) re-emission is considered. If additionally

- an isotropic source emission, e.g., $\Gamma_{src}(\boldsymbol{x}, \boldsymbol{\omega}) = E_{src}\frac{1}{\pi}\boldsymbol{n}_{\boldsymbol{x}} \cdot \boldsymbol{w}$,

- planar surface elements, e.g., triangles, and
- a constant re-emission factor $(1 - s)$ for each surface element

is assumed, (2.14) can be converted into the discrete radiosity equation (2.15).

$$A_i \cdot C_{in_i} = \sum_j^{N_\mathcal{P}} A_j \cdot E_{src_j} \underbrace{\frac{1}{\pi A_j} \int_{A_i} \int_{A_j} \frac{cos(\theta_{idA_j})}{r_{ij}^2} cos(\theta_{jdA_i}) dA_j dA_i}_{F_{ji}} + \\ + \sum_j^{N_\mathcal{S}} (1-s) A_j \cdot C_{in_j} \underbrace{\frac{1}{\pi A_j} \int_{A_i} \int_{A_j} \frac{cos(\theta_{idA_j})}{r_{ij}^2} cos(\theta_{jdA_i}) dA_j dA_i}_{F_{ji}} \tag{2.15}$$

$A_i \cdot C_{in_i}$ is the total arriving particle flux for element i, r_{ij} is the distance between two points on element i and element j, and θ_{jdA_i} denotes the angle between the surface normal of element i and the direction towards a point on element j. The view factors F_{ji} are identified; in the radiosity framework (which originates from the field of radiative heat transfer) a view factor is the portion of the total radiation of a surface patch j which is received by another surface patch i.

The radiosity formulation is the special (most simple) case of the "straightforward approach", when only diffuse emission and re-emissions are considered. When modeling non-diffuse effects the choice of the basis functions c_i must be appropriate to capture the properties of the arriving particle flux Γ_{in}, which are of importance during the extraction of the surface rates.

A linear system of equations is constructed from (2.15) in case of a radiosity setting and from (2.14) in the general case. If N_c basis functions are used, the size of the system matrix is $N_\mathcal{S} N_c \times N_\mathcal{S} N_c$. The number of non-zero entries in the matrix is $\mathcal{O}((N_\mathcal{S} N_c)^2)$ as potentially all surface elements are mutually visible. The implementation of boundary conditions (cf. Figure 2.1) leads to an extension of the system matrix with additional entries.

For highly resolved surfaces in a three-dimensional simulation, the direct integration method requires significant memory for storing the system matrix. From an implementation point of view, the selection of appropriate basis functions and the evaluation of resulting matrix elements is problematic [5]. During the evaluation of the matrix elements visibility tests between pairs of points on the surface are used to determine mutual direct visibility which is a condition for the exchange of particles. Some iterative methods offer a natural interpretation, when used to approximate a solution to the linear system, e.g., using $\mathbf{0}$ as initial guess and performing n iterations of the Jacobi method can be interpreted as only considering n re-emissions and omitting the influence of particles after the n-th re-emission; performing only one iteration corresponds to solely considering the direct particle flux from the source plane to the surface.

2.2.1.2 Bottom-Up Iterative Approach

Different from the method described above, the *bottom-up* iterative scheme does not generate a system matrix upfront. In the first iteration only the direct flux from the source plane $\mathcal{P}$ towards the surface $\mathcal{S}$ is considered using only the first summand of (2.4) leading to

$$C_1(\boldsymbol{x}) = \int_{\Omega} \Gamma_{in}(\boldsymbol{x}, \boldsymbol{\omega}_{d\Omega}) d\Omega = \int_{\Omega_{vis\mathcal{P}}} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{P}}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{d\mathcal{P}} \|^2} \left[\Gamma_{src}(\boldsymbol{x}_{\boldsymbol{d}\mathcal{P}}, -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{P}}})\right] d\Omega \ . \tag{2.16}$$

For a set of $N_{\mathcal{S}}$ points on the surface $\mathcal{S}$ the integral is evaluated numerically by means of a directional sampling of the sphere. Various schemes can be used to sample the spherical directions. Figure 2.2 illustrates the use of a latitude-longitude grid or geodesic grids in two dimensions. Visibility tests are necessary for each

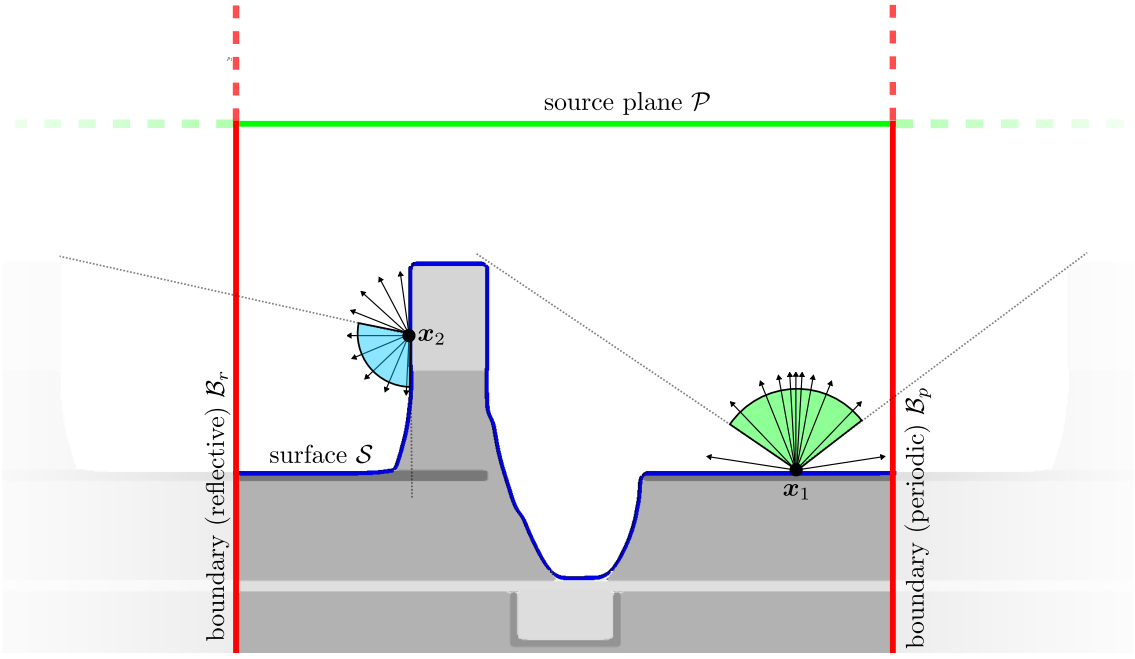


Figure 2.2: Schematic illustration of the *bottom-up* iterative method: Two different sampling schemes for the spherical directions are shown. For point $\boldsymbol{x}_1$ the visible solid angle of the source plane is shown and together with a latitude-longitude-like scheme for the sampling directions approximated for two-dimensions. For point $\boldsymbol{x}_2$ the visible solid angle of the surface is shown together with a geodesic-like scheme for the sampling directions approximated for two dimensions.

sampling direction to detect if the source plane is directly visible. A discretization of $\mathcal{P}$ is not necessary if Γ_{src} does not depend on the position on $\mathcal{P}$. To approximate non-diffuse re-emission in subsequent iterations certain averaged properties of the arriving particle flux can be captured and stored during the numerical integration, e.g., a mean incoming direction.

In all of the subsequent iterations the numerical integration is performed not over the visible directions towards the source plane $\Omega_{vis\mathcal{P}}$, but over the visible directions

towards the surface.

$$C_2(\boldsymbol{x}) = \int_{\Omega} \Gamma_{in}(\boldsymbol{x}, \boldsymbol{\omega}_{d\Omega}) d\Omega = \int_{\Omega_{vis\mathcal{S}}} \frac{\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{P}}} \cdot \boldsymbol{n}_{\boldsymbol{x}}}{\| \boldsymbol{x} - \boldsymbol{x}_{d\mathcal{S}} \|^2} \left[\Gamma_{re}(\boldsymbol{x}_{d\mathcal{S}}, -\boldsymbol{\omega}_{\boldsymbol{x}\boldsymbol{x}_{d\mathcal{S}}})\right] d\Omega \tag{2.17}$$

An intersection test is required to obtain the closest intersection with the surface $\mathcal{S}$ along the sampling direction.

The final surface rates are calculated using

$$R(\boldsymbol{x}) = C_1(\boldsymbol{x}) + C_2(\boldsymbol{x}) + ... + C_n(\boldsymbol{x}) \ . \tag{2.18}$$

The computational demands in terms of storage depend on the complexity of the captured properties of the arriving flux distribution for each integration point and are $\mathcal{O}(N_{\mathcal{S}} \cdot n)$. Depending on the number of iterations n and the chosen sampling scheme the necessary visibility/intersection tests are the predominant computational task. The integration schemes are free to change in each iteration, e.g., the schemes can be adopted to known properties of Γ_{src} in the first iteration by, e.g., using a grid refined towards a pole as shown in Figure 2.2. A boundary causes a reflection or translation of the direction of the visibility/intersection test for reflective and periodic conditions, respectively.

2.2.1.3 Top-Down Monte Carlo Approach

The *top-down* Monte Carlo sampling scheme launches a large number of particles N_p on the source plane $\mathcal{P}$ and traces their trajectories inside the domain. The position and direction of the particles starting on $\mathcal{P}$ are generated randomly but weighted so that they collectively resemble the properties of Γ_{src}. When a particle p first hits the surface $\mathcal{S}$ at location $\boldsymbol{x}$, it contributes with its properties (e.g., incoming direction, energy, and particle type) directly to the local surface rates. Re-emission is modeled by launching one or more new particles from the same location $\boldsymbol{x}$, where the original particle hit the surface. Again, the re-emitted particles are generated randomly but collectively resemble the re-emission distribution $\Gamma_{re}(\boldsymbol{x})$. Figure 2.3 illustrates the trajectories of 4 exemplary particles launched from the source plane through the domain.

Similar to the methods discussed above, a maximum number of re-emissions can be defined at which the generation of new particles is stopped. Alternatively each particle has a "weight" and a threshold value determines when a particle trajectory is terminated; the weight is updated after each reflection event of the particle. Due to the stochastic nature of the approach, noise is present in the resulting surface rates. The storage requirement for this method is low as for each particle hit the contribution to the rate is directly evaluated. Complex re-emission properties, e.g., specular-like reflections, can be considered straightforwardly [5].

To obtain the closest intersection with the surface for a given particle position and direction an intersection test computationally very similar to the visibility test is required.

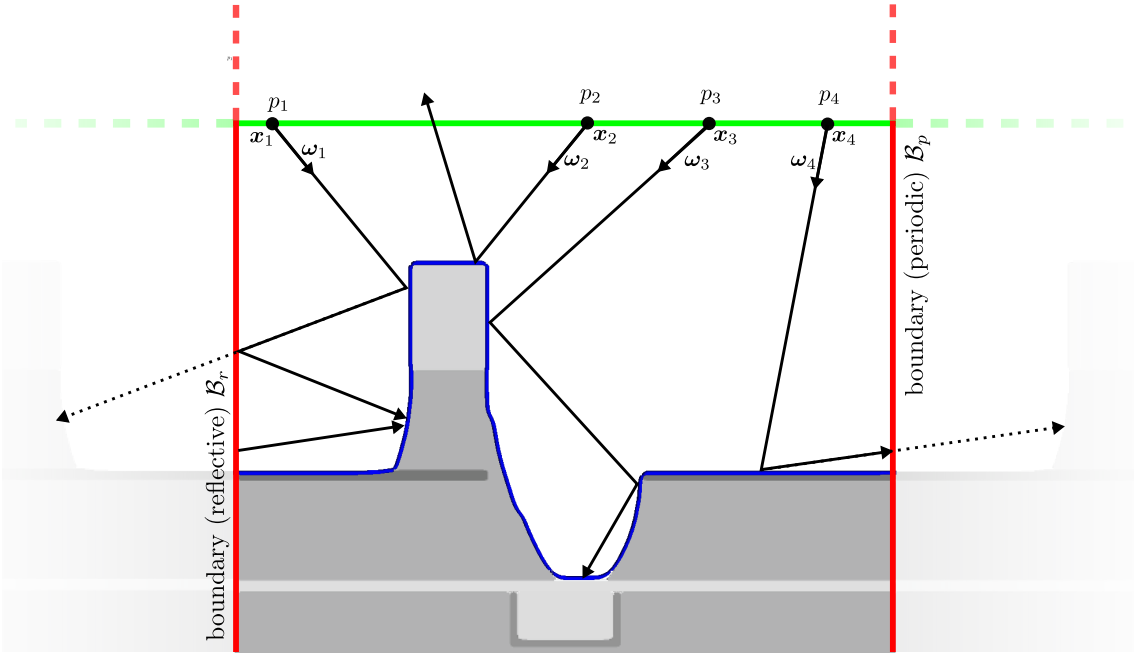


Figure 2.3: Schematic illustration of the *top-down* Monte Carlo method: The trajectories of 4 exemplary particles p_1 to p_4 are shown. The effect of reflective (left) and periodic (right) boundary conditions is shown.

2.2.2 Efficient Visibility/Intersection Tests

All methods described above rely on visibility/intersection tests. How dominant these tests are in relation to the overall computational task depends on the method:

- In the "straightforward approach" the visibility tests are used during the construction of the entries of the system matrix. The number of tests is $\mathcal{O}(N_\mathcal{S} \cdot N_\mathcal{S})$. For a surface discretized with 1 million elements $\approx$ 0.5 trillion $(10^6 \cdot (10^6 - 1)/2)$ visibility tests are necessary to evaluate the mutual visibility between each pair of elements. The surface normals can be used to reduce this number by first evaluating the sign of their scalar product, where only for a value < 0 the visibility test is performed.

- In the *bottom-up* iterative scheme visibility tests are used in the initial iteration and intersection tests are used in all subsequent iterations. The number of tests for n iterations is $\mathcal{O}(N_\mathcal{S} \cdot n)$, if a fixed number of sampling directions is used. For a surface discretized with 1 million elements and considering 1000 sampling directions the necessary number of tests is 1 billion $(10^6 \cdot 1000)$ per iteration.

- In the *top-down* Monte Carlo scheme the number of tests equals the number of particles launched from the source plane N_p multiplied with the average number of re-emissions per particle n_{re}. The number of tests is $\mathcal{O}(N_p \cdot n_{re})$. For a surface discretized with 1 million elements and aiming at an average

number of 1000 particles (for direct flux) reaching each element, 1 billion ($10^6 \cdot 1000$) of intersection tests are necessary for the direct flux.

The considerations above show clearly that the first method is not right away suited for high-resolution three-dimensional simulations. The comparison between the other two methods reveals a similar number of necessary intersection tests per re-emission. Regardless of the approach, a computationally efficient implementation of a visibility/intersection test is desirable as a vast number of tests are used in all of the methods described above, especially if re-emissions are considered.

2.2.2.1 Ray-Surface Intersection

A visibility/intersection test is a ray-surface intersection test for a representation of the surface consisting of N primitives, a starting point of the ray (origin) and a search direction. A straightforward implementation is to test each primitive for an intersection along the search direction and to track the minimum intersection distance. The result is the closest intersection point on one of the primitives. Due to its complexity $\mathcal{O}(N_{primitives} \cdot N_{tests})$ such a straightforward implementation is prohibitive considering the required number of tests mentioned above.

In the field of computer graphics, ray-surface intersection tests are used intensively for ray tracing [22] as a rendering technique. Specialized frameworks are used to perform these intersection tests (ray casts) efficiently. All have in common that initially a tree-like acceleration structure is constructed from the scene, e.g., a bounding volume hierarchy (BVH). This allows to traverse a ray more efficiently through the domain without performing intersection tests with all primitives but mostly with bounding volumes of the BVH. Typically, these frameworks are optimized to perform tasks occurring in rendering situations, e.g., to traverse a bundle of spatially coherent rays (i.e., rays with similar direction) together through the scene. The applied performance metric is million ray casts per second (Mrays/s). Non-coherent ray casting, as required for the visibility/intersection tests during the particle transport, is typically also implemented very efficiently, although the maximum Mray/s scores are reached for ray bundles.

The following selection of frameworks perform accelerated ray casting using explicit surfaces, e.g., triangle meshes:

Embree [23][24] is an open-source collection of C++ ray tracing kernels developed and maintained by *Intel*. The kernels are optimized by making use of SIMD (single instruction, multiple data) instruction sets available on modern central processing units (CPUs). It includes optimized algorithms for coherent and incoherent ray workloads. All kernels are solely implemented using a single-precision floating point representations.

NanoRT [25] is an open-source C++ header only CPU ray tracing kernel. It supports ray tracing using single- and double-precision floating point representation.

Optix Prime [26] is a closed-source C++ ray tracing library distributed by *Nvidia*. It is optimized for ray-triangle intersections on graphics processing units

(GPUs) and solely uses single-precision floating point representation. It includes fallback implementations for CPUs, if no supported GPU is found on the system. It depends on CUDA [27].

pbrt-v3 [28] is an open-source C++ ray tracing renderer, accompanying the book *Physically Based Rendering: From Theory to Implementation* [21]. It supports ray tracing using single- and double-precision floating point representation.

RadeonRays [29] is an open-source C++ ray intersection library maintained by *AMD*. It is targeted at CPUs and GPUs. The ray tracing is implemented using the OpenCL 1.2 standard.

All of the above mentioned frameworks support triangular meshes. The construction of an acceleration structure (i.e., an optimized construction of a BVH and a data structure to store and access the BVH) is provided by all frameworks.

In volume rendering application, rays are casted into a scene consisting of implicit surfaces, e.g., signed-distance functions. Similar to ray casting for explicit surfaces, an acceleration structure, e.g., a BVH, is used for efficient traversal of the ray. The actual intersection with the surface is obtained by marching the ray back and forth, until an intersection (i.e., change of sign) is detected. Sphere tracing [30] is a ray marching algorithm using the signed-distance information to determine the next step length along the search direction. Concerning data structures in this context, there is only one framework which stands out:

OpenVDB [31] is an open-source C++ project developed and maintained by *DreamWorks Animation*. It provides a data structure for sparse volume data bundled with a large toolset to operate on the sparse volume data. The data structure is a tree-like structure of 4 levels with non-equal branching factors. It implements ray marching using the sparse volume data structure directly without constructing a BVH [32].

2.3 Surface Evolution Using the Level-Set Method

The normal surface velocity V_n is prescribed by the surface velocity model (2.1) and is available for a set of points on the surface. The objective is to evolve the material surfaces according to V_n. As the evolution is unconstrained the surfaces may undergo topological changes, e.g.,

- a material surface vanishes completely, or
- a portion of a surface is pinched off, or
- two surfaces merge into one larger surface.

The majority of the simulators presented in Section 2.1 relies on a level-set-based method to evolve the surface as it offers robust treatment of the topological changes mentioned above. Before more details on the level-set method are presented, the three main groups of techniques used to model surface evolution are briefly explained and advantages and disadvantages are discussed.

Explicit Representation. The surface is discretized as an explicit set of nodes with connectivity information, e.g., a triangle mesh. To evolve the surface each node is advanced according to the normal velocity. Initially equidistantly spaced nodes can accumulate or rarefy depending on the curvature and surface velocity. To ensure a maximal resolution (maximal distance between nodes) and to avoid numerical instabilities in accumulated regions, nodes must be redistributed; if the surface area is increasing, new nodes must be inserted and connected. An unconstrained advancement of the nodes leads to a self-intersection of the elements. These self-intersections must be healed/resolved from time to time to ensure a valid surface representation. In the case of a topological change, a change of the connectivity between nodes is required, which is not easily implemented robustly. In general, there is no restriction for the time step size but larger time steps (i.e., larger deformations) make it more difficult to find robust solutions for the redistribution, intersection-healing, and connectivity adjustments mentioned above. However, sharp features of the surface are preserved well. In [33] a method for tracking fluid surfaces using a triangle mesh is presented. In [34] and [35] algorithms for adaptive restructuring of meshes on evolving surfaces are presented.

Cellular (Voxel) Representation. The surface is represented by a Cartesian grid of cubic cells. Numerical values (typically a scalar value in $[0, 1]$) are assigned to each cell identifying the cell as interior, boundary, or exterior cell. The surface is implicitly represented through the boundary cells. The numerical values are updated using local update rules incorporating the values of neighboring cells and the surface velocity. The maximum time step is enforced by the update rules. Topological changes are handled robustly. Features below the resolution of the cubic cells cannot be represented. In [36] a cellular material representation is used to evolve the surface in a plasma etching simulation. The determination of the surface normal requires averaging across several cells [36]. In [37] a computationally efficient three-dimensional cellular automata model is used to model the surface evolution in an etching simulation. *SEmulator3D* [12] uses a voxel modeling approach for geometric processing steps.

Level-Set-Based Representation. The surface is represented implicitly by the zero-level-set of a scalar field. The zero-level-set is the zero-isoline of a two-dimensional scalar field or the zero-isosurface for a three-dimensional scalar field. An evolution of the implicit surface is therefore equivalent to the advection of the scalar field. The equation for this advection is the level-set equation. The numerical solution is typically performed using a Cartesian grid combined with appropriate finite difference schemes for the spatial and temporal derivatives. The time step is limited by the temporal discretization scheme. Topological changes are inherently supported. In [38] the level-set method is mathematically derived, related methods are covered and numerous application examples are shown. The method is continuously being improved, e.g., to improve mass conservation and unintended smoothing of the

surface [39][40][41][42]. The surface evolution in *Victory Process* [8], *ViennaTS* [14], and *Sentuarus3D* [10] is based on the level-set method.

An overview of surface tracking approaches (including the ones mentioned above) in the field of fluid dynamics is provided in [43].

In the following, details on the level-set method including velocity extension, normalization, discretization, surface extraction, and data structures are provided.

2.3.1 The Level-Set Method

The level-set method takes the approach that the surface/interface to be tracked is not explicitly represented. In the following a simple example to illustrate the method is considered: A spherical surface shrinking with unit speed.

The surface of a sphere of radius $r = 1$ and center at $c = [0, 0, 0]$ is represented with a three-dimensional scalar level-set function ϕ, where the values are initialized as $\phi(\boldsymbol{x}) = \parallel \boldsymbol{x} \parallel -1$. Using this initialization $\phi = 0$ on the surface, $\phi < 0$ inside, and $\phi > 0$ outside the sphere. If a vector velocity field is defined as $\boldsymbol{V}(\boldsymbol{x}) = \frac{-\boldsymbol{x}}{\|\boldsymbol{x}\|}$, i.e., an inward pointing unit speed, and the level-set function is advected using

$$\frac{\partial \phi(\mathbf{x})}{\partial t} = - \underbrace{\boldsymbol{V}(\boldsymbol{x}) \cdot \frac{\nabla \phi(\mathbf{x})}{\parallel \nabla \phi(\boldsymbol{x}) \parallel}}_{V_{n_{\nabla\phi}}(\nabla\phi, \boldsymbol{x})} \parallel \nabla \phi(\mathbf{x}) \parallel , \tag{2.19}$$

the implicit surface (represented by the zero-isosurface of ϕ) is a sphere shrinking with unit speed. Equation (2.19) relates the change of ϕ over time to the norm of the gradient $\parallel \nabla\phi \parallel$ weighted with a speed function $\boldsymbol{V}$ projected into the direction of the gradient. For time $t > 1.0$ the zero-isosurface of ϕ vanishes (and so does the tracked interface) as all values of ϕ are now larger than zero. Renaming the velocity into the direction of the gradient $V_{n_{\nabla\phi}}(\nabla\phi, \boldsymbol{x})$ to $F(\nabla\phi, \boldsymbol{x})$ (2.19) can be rewritten as

$$\frac{\partial \phi(\mathbf{x})}{\partial t} + F(\nabla\phi, \boldsymbol{x}) \parallel \nabla \phi(\mathbf{x}) \parallel = 0 , \tag{2.20}$$

which is the standard form of the level-set equation. The dependencies of F determine if (2.20) is linear or non-linear:

- (2.20) is linear if F is independent of ϕ and its derivatives and only depends on the location $\boldsymbol{x}$, e.g., the linear transport equation $\phi_t + F\phi_x = 0$ (for one dimension).
- (2.20) becomes non-linear if F is dependent on ϕ or $\nabla\phi$, e.g., $F = \phi$ leading to $\phi_t + \phi\phi_x = 0$ (i.e., the *inviscid Burgers' equation* for one dimension [44]).

In the case of a process simulation, a meaningful velocity is only available for the points on the surface, but a velocity field F is required for the advection of the level-set function ϕ. The next section describes how a suitable velocity field is obtained from the surface velocities using extrapolation.

2.3.2 Velocity Extension

In a process simulation, the velocity field has no physical meaning as velocities are only meaningful on the surface itself. There are generally no requirements for the velocity field F besides it must agree with the surface velocity at the zero-level-set which corresponds to the interface. The most straightforward approach is to extrapolate from the closest point on the surface $\mathcal{S}$ [45]: For a point $\boldsymbol{x}$ away from the surface the velocity from the closest point on the surface $\boldsymbol{x}_{cp} \in \mathcal{S}$ is used.

Depending on the data structure, the identification of the closest point on the surface can be a demanding task. In [38] an alternative approach for obtaining the velocity field is described, where the velocity field is "marched away" from the surface with the fast marching method [46] to solve the static boundary value problem

$$\| \nabla\phi(\boldsymbol{x}) \| F(\boldsymbol{x}) = 1, \quad \phi(\boldsymbol{x}) = 0 \text{ on } \Gamma \,, \tag{2.21}$$

where Γ is the interface and F is a strictly positive speed function only depending on the location $\boldsymbol{x}$.

A consequence of how the velocity is extended is how well the level-set function retains a signed-distance function over time. While this property is generally not maintained using the approach in [45], the alternative approach [38] analytically maintains the signed-distance property but discretization errors in ϕ and its derivatives will gradually destroy this property over time. The next section describes how the signed-distance property of the level-set function can be sustained.

2.3.3 Sustaining a Signed-Distance Property

In the example of the spherical surface in Section 2.3.1 the level-set function ϕ has been initialized to a signed-distance function. That is, the magnitude $|\phi|$ encodes the distance to the zero-level-set and the sign encodes the inside $(-)$ and outside $(+)$ regions. The signed-distance property is not maintained as ϕ evolves in time due to the construction of the extended velocity field itself or due to discretization errors. It is desirable to maintain a signed-distance property of ϕ:

- If ϕ is very "flat" or very "steep" the numerical determination of the zero-level-set locations is less accurate.
- If $|\nabla\phi|$ varies the accuracy of numerical derivatives suffers [47]; this includes the accuracy of the curvature and the surface normal.
- The closest point on the surface can be approximated by a scaled step along the gradient: $\boldsymbol{x}_{cp} = \boldsymbol{x} - \phi(\boldsymbol{x})\nabla\phi(\boldsymbol{x})$.
- The level-set function qualifies to robustly apply constructive solid geometry (CSG) operations and geometrical "offset" operations [48].

For the reasons given above most level-set applications perform a process of "reshaping" the level-set function in regular intervals with the aim to restore the signed-distance property. This process if often called *reinitialization*, *redistancing*, or *normalization*. Related methods can be categorized into two distinct groups:

Boundary Value Problem-based methods solve (2.21). Examples are the fast marching method [46], the fast sweeping method [49], and the fast iterative method [48]. Solutions to the boundary value problem can be generated with near optimal complexity. The solution near the interface is used as initial condition and is not varied, which is a desirable property. On the other hand, deviations from the signed-distance property near the interface are not corrected.

Flow-based methods solve an equation of the form (2.22). The flow defined by (2.22) has normalizing properties, e.g., it vanishes when the signed-distance property is obtained. The solutions near the interface are also influenced by these methods with the benefit that deviations from the signed-distance function near the interface are corrected. A consequence is that the interface is also slightly advected by the flow.

$$\begin{aligned} \frac{\partial \phi(\mathbf{x})}{\partial t} + \sigma(\boldsymbol{x}) \parallel \nabla \phi(\mathbf{x}) - 1 \parallel = 0 \\ \sigma(\boldsymbol{x}) = \operatorname{sgn}(\phi(\boldsymbol{x})) = \frac{\phi(\boldsymbol{x})}{\sqrt{\phi(\boldsymbol{x})^2 + \epsilon}} \end{aligned} \tag{2.22}$$

A discussion of advantages and disadvantages for boundary value problem-based or flow-based methods for normalization can be found in [47].

2.3.4 Discretization

The level-set equation (2.20) belongs to the class of Hamilton-Jacobi equations of the form

$$\begin{aligned} \frac{\partial \phi(\mathbf{x})}{\partial t} + H(\boldsymbol{x}, t, \phi, \nabla\phi) = 0, \quad \text{with the Hamiltonian} \\ H(\boldsymbol{x}, t, \phi, \nabla\phi) = F(\nabla\phi, \boldsymbol{x}) \parallel \nabla \phi(\mathbf{x}) \parallel . \end{aligned} \tag{2.23}$$

On uniform Cartesian grids a set of discretization schemes has established itself as a widely used default choice [7].

2.3.4.1 Spatial Derivatives

The weighted ENO (essentially non-oscillatory) scheme [50][51], in short WENO, is used for the approximation of the first derivatives of ϕ. The WENO scheme is based on the ENO scheme [52], which adaptively selects a finite difference stencil (out of a fixed set of stencils) for the approximation: By calculating the smoothness of ϕ inside each stencil's interval, the selected stencil is the one the interval of which does least overlap with a discontinuity of ϕ. The WENO scheme does not solely use one stencil but instead uses a weighted convex combination of all stencils, where the weights depend on a smoothness measure of ϕ in the stencil's interval.

2.3.4.2 Hamiltonian Discretization

The level-set equation (2.20) can be related to a scalar conservation law. Starting with the one-dimensional form of (2.20)

$$\phi_t + H(\phi_x) = 0 \ , \tag{2.24}$$

and $\phi_x = p$, and differentiating results in

$$\left[\left[\int p dx\right]_t\right]_x + \left[H(p)\right]_x = 0 \tag{2.25}$$

which can be simplified to the conservation law

$$p_t + \left[H(p)\right]_x = 0. \tag{2.26}$$

If $H(p) = \alpha p$, this gives the linear transport equation $p_t + \alpha p_x = 0$ and upwind finite difference schemes, which take the derivatives in the direction of information propagation, can be used, as the upwind direction is known beforehand. For a nonlinear conservation law, the transport depends on the solution itself and the information propagation direction cannot be determined a priori. Furthermore, finite difference schemes are not well suited for scalar conservation laws as solutions are not necessarily smooth. The integral form of the conservation law

$$\left[\int_a^b p dx\right]_t = -\int_a^b \left[H(p)\right]_x dx = -\left[H(p(b)) - H(p(a))\right] \tag{2.27}$$

is therefore the starting point for the numerical schemes for scalar conservation laws. Using a finite volume approach, the cell averages at the next time step are obtained by integrating (2.27) over the domain $[x_{i-1/2}, x_{i+1/2}] \times [t^n, t^{n+1}]$ leading to

$$(p_i^{n+1} - p_i^n)\Delta x = -\Delta t(\bar{G}_{i+1/2}^n - \bar{G}_{i-1/2}^n) \ , \tag{2.28}$$

where $\bar{G}$ are the fluxes at the cell boundaries

$$\bar{G}_{i+1/2}^n = \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} H(p(x_{i+1/2}, t))\ dt \ , \tag{2.29}$$

which are not known a priori. The Godunov method [7][53][54] provides an a priori explicit approximation to $\bar{G} = G$ by identifying a *Riemann problem* for each cell interface.

Different schemes exist [55] to approximate a solution to the Riemann problems, e.g., the Lax-Friedrich scheme, the Engquist-Osher scheme, and the Godunov scheme (not to be confuse with the Godunov *method* above). Each of the schemes produces an approximation $G(p_{i+1/2}) \approx g(p_i, p_{i+1})$ which results in

$$\frac{p_i^{n+1} - p_i^n}{\Delta t} = -\frac{g(p_i^n, p_{i+1^n}) - g(p_{i-1}^n, p_i^n)}{\Delta x} \ . \tag{2.30}$$

The approximation schemes for H in (2.24) rely on the numerical fluxes $g(p, p)$. Using $\phi_x = p$ equation (2.24) can be rewritten as

$$\phi_t = -H(\phi_x) = -H(p) \ , \tag{2.31}$$

and an approximation of $H(p)$ is given by the numerical flux function $g(p, p)$. This leads to

$$H(p_i^n) \approx g(p_{i-1/2}, p_{i+1/2}) \ , \tag{2.32}$$

and finally (for an explicit Euler step) to

$$\frac{\phi_i^{t+1} - \phi_i^t}{\Delta t} = -g(p_{i-1/2}, p_{i+1/2}). \tag{2.33}$$

2.3.4.3 Temporal Discretization

Total Variation Diminishing Runge-Kutta (TVD-RK) schemes [56][57] are typically used to discretize (2.20) in time [7]. The first-order accurate scheme (TVD-RK1) corresponds to a single explicit Euler step, i.e., for one dimension

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + g^{GODUNOV}(\phi_{x_{i-1/2}}^{WENO-}, \phi_{x_{i+1/2}}^{WENO+}) = 0 \ , \tag{2.34}$$

where the Godunov scheme is used for the numerical flux and the WENO scheme for the derivatives of ϕ at $i - 1/2$ and $i + 1/2$. Second (TVD-RK2) and third order (TVD-RK3) schemes combine multiple temporary Euler steps to obtain higher order accuracy.

The size for the discrete time step is limited by the assumptions of the spatial discretization scheme and the explicit time integration scheme. To obtain a stable scheme, the maximum time step is

$$\Delta t < \frac{\Delta h}{|F_{max}|} \cdot \alpha_{CFL} \ , \tag{2.35}$$

where Δh is the spatial grid spacing, F_{max} is the maximum value in the velocity field, and $\alpha_{CFL} \in \ ]0, 1]$ is a positive constant depending on the discretization schemes. This condition is known as the Courant-Friedrichs-Lewy (CFL) condition [58].

2.3.5 Data Structures

The straightforward implementation of the level-set method using a multi-dimensional dense array of values leads to storage complexity $\mathcal{O}(n^d)$ where n is the average spatial resolution and d is the number of spatial dimensions. For high-resolution three-dimensional simulations a dense array is not practical, e.g., a domain of 1000^3 requires more than 7 GB of memory, if double-precision floating values are stored. The memory footprint can be reduced by using single-precision or even less accurate floating point representations.

To overcome the large memory footprint an attractive approach is to only store the level-set function values near the surface. This is feasible as it is sufficient to only solve the level-set equation inside this *narrow band* around the surface which solely defines the position of the implicit surface [59][60]. Data structures can be distinguished into two main groups:

- Tree-based approaches use a tree of nested cubes, where the leaf nodes store the level-set function values [61][62]. The non-leaf nodes contain either only the sign or level-set values at a reduced resolution.
- Run-length encoded approaches compress regions away from the narrow band by just storing a sign denoting inside or outside [5][63][64].

General requirements for the data structures are an efficient mechanism to adopt the narrow band after the surface has evolved, and fast access methods for sequential and random access.

2.3.6 Surface Extraction

An explicit representation of the surface is obtained by performing an isosurface extraction. The extraction algorithms can be categorized into the following groups [65]:

Primal methods produce a polygonal mesh by connecting the intersection points of the isosurface with a structured grid. The *marching cubes* algorithm is robust, straightforward to implement, and is the standard algorithm for surface extraction from a signed-distance field. Variations of the original algorithm exist, e.g., to improve mesh quality and unwanted smoothing of sharp features [65][66].

Dual methods identify a representative point for each cell of the structured grid and connect these points to form a polygonal mesh. The *dual marching cubes* [67] is an example for a dual extraction method. In [68], an algorithm for dual contouring is introduced and compared to primal methods.

2.4 Summary

Most process simulators rely on the level-set method to evolve the surface due to its inherent support for arbitrary surface evolutions. An implicit representation of the surface (level-set function) is maintained throughout the simulation. For efficiency reasons, the level-set function is stored using sparse volume data structures which only store the values inside the narrow band around the zero-level-set. The fact that it is sufficient to solve the level-set equation only in the vicinity of the zero-level-set makes this memory efficient approach possible.

The particle transport is an integral part of any advanced simulation of etching or deposition processes, as the surface rates (entering the surface velocity model) depend on the particle distributions inside the domain. A vast number of visibility/intersection tests are a major computational task in all presented methods to

model the particle transport. The particle transport is calculated for every time step of a simulation.

The visibility/intersection tests can either be performed using an implicit representation (level-set function) or using an explicit representation of the the surface. The explicit representation is created from the level-set function using an extraction algorithm.

Visibility/intersection tests are ray-surface intersection tests (ray casting). Ray tracing is a rendering technique used in the field of computer graphics, which relies on casting (light) rays into the scene. Numerous optimized libraries implementing efficient ray casting are available, originally targeted for rendering applications. Most of those libraries perform all computations using single-precision floating point operations.

Starting out with the approaches for the particle transport described above, the aim of this work is to reduce the computational workload contributed by the particle transport calculation during a process simulation. The next chapter introduces the simulation framework which is used to implement and validate the methods presented in the remainder of this work.

Chapter 3

Simulation Platform

The platform introduced in this chapter is used as a development platform for implementing and evaluating novel computationally efficient approaches to calculate the particle transport in three-dimensional etching and deposition simulations. The approaches presented in Chapters 4, 5, and 6 were implemented and validated using this platform.

The performance of a particle transport calculation can be investigated detached from a simulation of a dynamic surface, i.e., using

- a set of surface geometries with corresponding surface models,
- information about the domain and boundary conditions, and
- information about the properties of the particle sources.

However, there is a disadvantage when using such a static setup: The accumulated effect of the particle transport method on the final topography at the end of the simulation cannot be assessed as the surface is not advected.

The alternative is to embed the evaluation directly into a process simulator. This provides access to all simulation results and allows for the evaluation of particle transport methods throughout the course of a full simulation. However, most available process simulators (cf. Section 2.1) are closed-source projects and therefore unsuitable. At the time of writing, *ViennaTS* [14] is the only available three-dimensional fully open-source process simulator. Due to a lack of functionality in *ViennaTS* regarding geometry construction, serialization of the level-sets, and surface extraction a standalone evaluation platform is desired.

To provide an overview, Figure 3.1 illustrates the sequence of computational tasks during a process simulation using such a simulation platform: First, the domain and the layers (i.e., the level-set functions which delimit the material regions) are created (left column). Then, the main computational tasks in each time step of the simulation (middle column) are

- the extraction of an explicit representation of the *top layer*[1], i.e., a triangle mesh including the information of the active material for each triangle,

[1] See Section 3.1.2 for the definition of *top layer*.

- the calculation of the surface rates for each point on the *top layer*, which requires the calculation of the particle transport,
- the calculation of the surface velocities, i.e., the velocity for each point of the *top layer*,
- the extension of the surface velocities, i.e., preparing a velocity field for all level-set grid points in the narrow band of the *top layer*, and
- the advection of the *top layer* according to the surface velocity field including an appropriate update of the other layers (cf. Section 3.1.2).

Finally, the resulting layers and material regions are extracted and saved (right column).

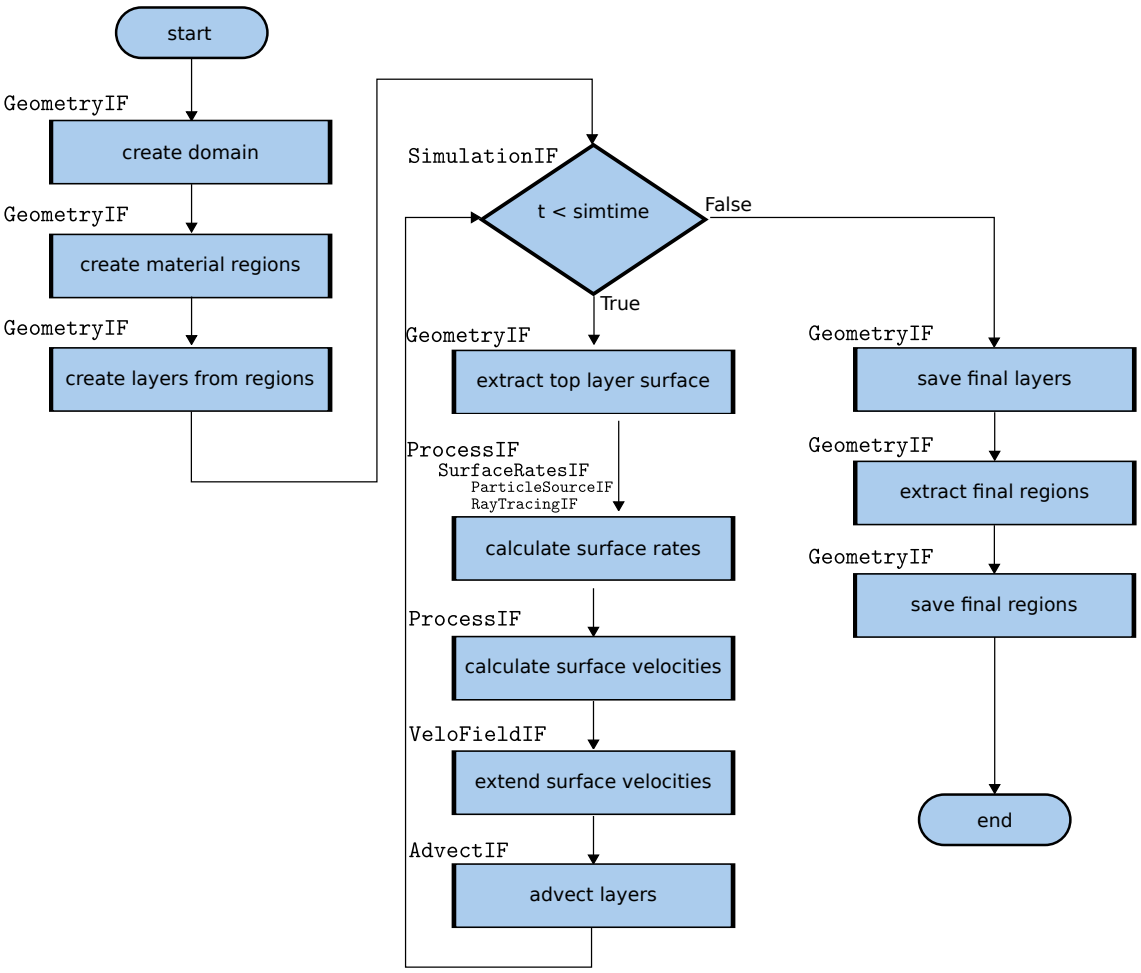


Figure 3.1: Coarse-grained flow chart of the process simulation identifying the main computational tasks: Surface extraction, surface rate calculation, surface velocity calculation, creating of the surface velocity field, and the advection of the layers. The relation of the computational tasks encapsulated by the interface classes (cf. Section 3.2) is indicated.

To minimize development effort, open-source third-party libraries were employed. Briefly, the simulation platform is based on *OpenVDB* [31], using its sparse volume

data structure; the tools provided by *OpenVDB* to manipulate the sparse data (e.g., narrow band level-set methods for advecting/normalizing level-sets and extraction of explicit surfaces) are also utilized. The ray-surface intersection queries for the calculation of the particle transport are performed using *Embree* [24] as external ray tracing library. Other ray tracing libraries (cf. Chapter 5) were used for validation and performance evaluations.

In the following, first, the necessary extensions to match the requirements of a feature-scale process simulation are discussed in detail. Then, an overview of the software architecture is given by focusing on the abstract interface classes which decouple the individual computational tasks. Finally, test cases and benchmarks are presented.

3.1 Requirements for Feature-Scale Process Simulation

The sparse volume data structure provided by *OpenVDB* [31] is used for surface representation. In [62] a detailed overview of the hierarchical data structure and implemented algorithms is provided and performance benchmarks are presented. *OpenVDB* suits the requirements for a level-set-based feature-scale process simulator to large extends as it provides [62]

- a configurable sparse data structure for volumetric data with cache efficient sequential access and fast random access (which is used to store the layers),
- a multi-threaded narrow band level-set method using TVD-RK explicit time integration schemes and a Godunov scheme combined with WENO schemes for spatial derivatives (which is used for the advection of the *top layer*),
- multi-threaded flow-based re-normalization of the narrow band using the same discretization schemes as for the advection (which is used for normalization of the *top layer* after each advection),
- geometry construction and multi-threaded CSG operations (which are used to update the other layers after the *top layer* is advected (cf. Section 3.1.2) and to construct the initial geometries),
- ray marching (i.e., ray casting) using the hierarchical volume data structure directly (which is used as a baseline for volume/implicit ray tracing performance, cf. Section 5.2),
- a fast dual method for the extraction of quad/triangle meshes (which is used for the extraction of a triangle mesh from the *top layer*, cf. Section 5.2),
- a conversion from quad/triangle meshes to volumes/level-sets (which is used to import explicit geometries), and

- serialization/deserialization of volumes/level-sets using a memory efficient file format (which is used to store intermediate and final results of the layers and regions).

Nevertheless, *OpenVDB* also lacks some requirements for a feature-scale process simulator, namely

- a confined domain size with prescribed boundary conditions (which is necessary when simulating only a small region of the wafer surface),
- a multi-material level-set advection logic (which is necessary if more than one material region is present in the simulation domain),
- a method for surface velocity extension (which is necessary as the surface velocities in a process simulation are only available for the surface, cf. Section 2.3.2), and
- a ray-surface intersection test which considers the confined domain and its boundary conditions.

In the following, the necessary extensions are briefly introduced.

3.1.1 Boundary Conditions

In feature-scale process simulations only a small region of the wafer surface is considered. Therefore, it is necessary to define boundary conditions for the vertical and lateral boundaries of the domain. The spatial position of the vertical boundary is adopted after each time step according to the new maximum vertical extents of the *top layer*. Two types of domain boundaries are commonly used for the lateral boundaries in a *Process TCAD* simulation: Periodic and reflective boundary conditions. Figure 3.2 illustrates the effect of periodic boundaries $\mathcal{B}_p$ (right) and reflective boundaries $\mathcal{B}_r$ (left) in two dimensions.

Reflective boundary conditions are possible for all geometries; periodic boundary conditions require that the initial geometry on one boundary matches the geometry on the opposing boundary for each applied dimension in which the boundary condition is applied.

If reflective boundary conditions are used, the emission of the particle sources must be rotationally symmetric with respect to the vertical axis; for periodic boundary conditions, the particle sources do not have this constraint.

The boundary conditions are implemented using a straightforward *ghost region* approach: The lateral boundaries of the simulation domain are extended by a thin layer of grid cells which are initialized according to the applied boundary condition. The surface advection is performed also for the *ghost regions*. The values in the *ghost regions* are discarded and re-initialized after each time step of the simulation. Figure 3.3 illustrates the *top layer* and *ghost regions* (blue) for a three-dimensional structure before and after a tilted directional etching (i.e., a non-symmetric particle source). Figure 3.4a illustrates the effective surrounding geometry for the initial

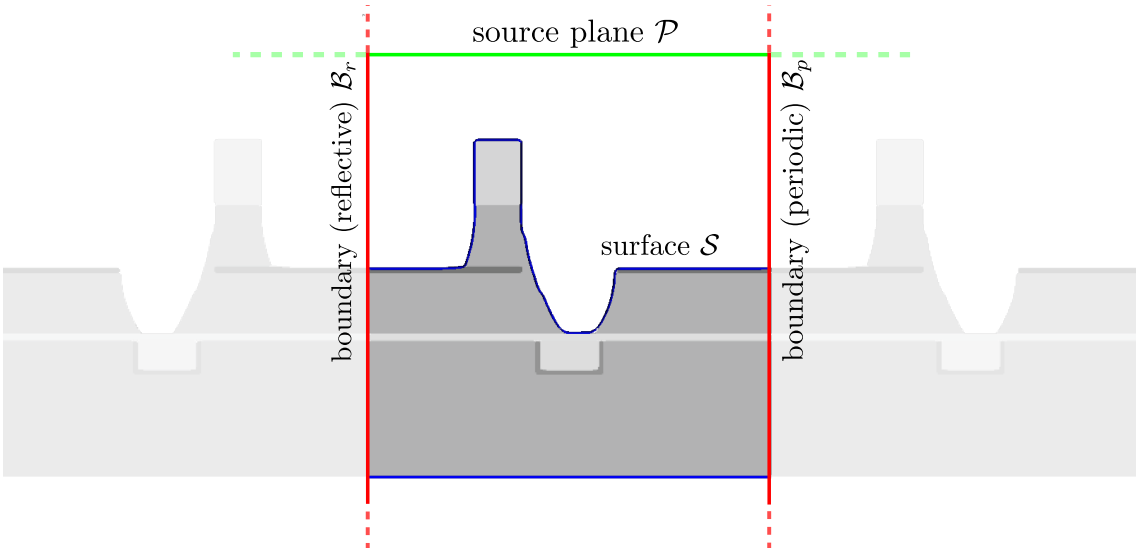
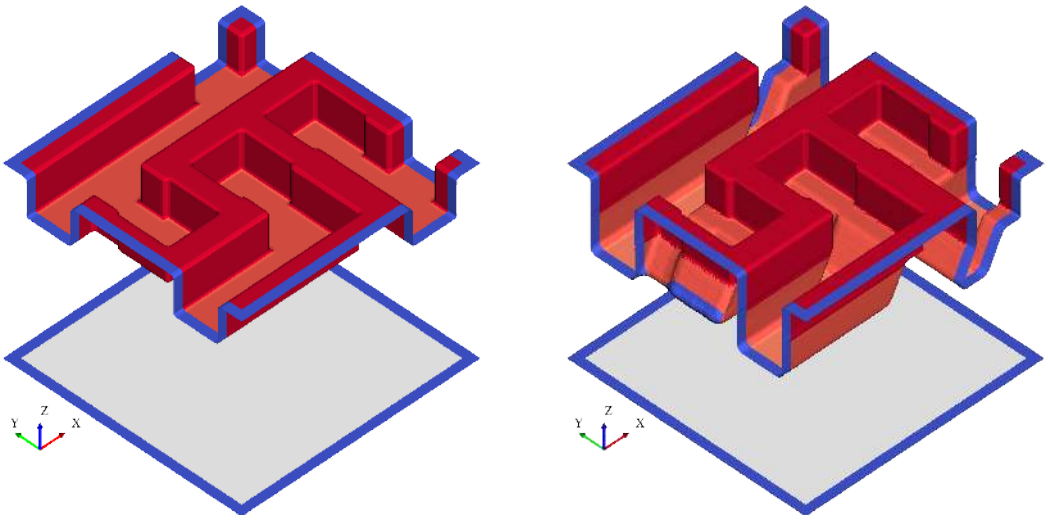


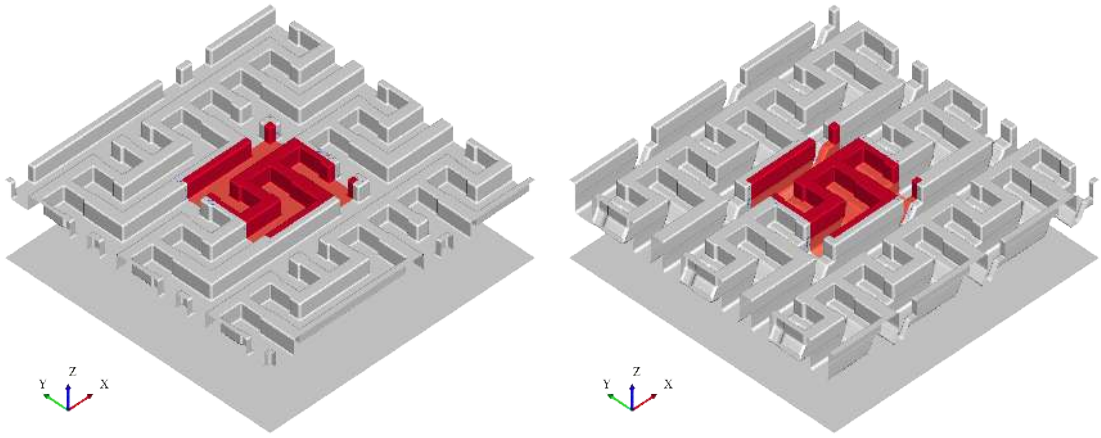
Figure 3.2: Two-dimensional illustration of a simulation domain with a reflective boundary condition $\mathcal{B}_r$ on the left side and a periodic boundary condition $\mathcal{B}_p$ on the right side. The vertical domain boundaries are infinite, i.e., are adopted according to the maximum extents of the *top layer* surface $\mathcal{S}$ (blue); the source plane $\mathcal{P}$ (green) is positioned according to the upper vertical domain extent.



(a) Initial *top layer* and *ghost region* **(b)** Result of tilted unidirectional etching

Figure 3.3: *Top layer* and *ghost region* of multi-material unidirectional tilted etching test case with periodic boundary conditions. (a) *Top layer* of initial geometry where the color indicates the active material and blue is the *ghost region*. (b) Resulting *top layer* and *ghost region* for unidirectional etching in direction $[-0.5, 0, -1]$.

geometry of Figure 3.3a, if periodic and reflective boundary conditions are combined. Figure 3.4b illustrates the same simulation results as Figure 3.3b but with the effective surrounding geometry which is produced by the periodic boundary conditions.



(a) Reflective (x-axis) and periodic (y-axis) boundary conditions

(b) Periodic boundary conditions

Figure 3.4: Illustration of the effective surrounding geometry for (a) the same geometry as Figure 3.3a but using combined reflective and periodic boundary conditions, and (b) the unidirectional etching test case with periodic boundary conditions (cf. Figure 3.3b).

3.1.2 Multi-Material Advection

Typically, different material regions are present during an etching process simulation, e.g., a substrate patterned with a mask. Also for deposition processes at least two materials are present, i.e., the initial material and the material which is deposited. The straightforward approach is to represent each material region with a corresponding level-set function. To simultaneously advect all material regions, each region would then be advected separately, leading to potentially mutual penetration. In this case, the parts of a region which are penetrated by another region would be treated inactive, i.e., not subject to advection. One approach would be to perform a Boolean operations between material regions to dissolve the penetrations; in this case a strategy to decide which material fills the former penetrated volume has to be set up.

Another approach [5][14] is to not advect each material region separately but to construct a total union of all regions and advect this *top layer*. To be able to advect the *top layer* with the correct surface speed of the underlying material region, it is necessary to detect the active material for each point on the *top layer*. This active material for a point $\boldsymbol{x}$ on the *top layer* is obtained by querying the value of all level-sets at $\boldsymbol{x}$: The construction of the *top layer* implies that the values of the queries are all ≤ 0. The material of the level-set with the smallest value is considered active; the active material at a point on the surface is therefore

$$M(\boldsymbol{x}) = \min_{m \in \{1..N_m\}} \phi_m(\boldsymbol{x}) \ , \tag{3.1}$$

where N_m is the number of materials and ϕ_m is the level-set function corresponding

to material m. Additionally, the level-sets have a fixed order and the lower level-set is chosen as active material, if the values are numerically identical.

However, with this approach it is not straightforward to deposit (i.e., the surface velocity $V_n > 0$) multiple materials simultaneously; instead one is restricted to deposit only one "top material" where $V_n > 0$. If $V_n < 0$, i.e., material is removed, the *top layer* is advected according to the underlying active material. In a subsequent Boolean operation between the advected *top layer* and each material region, the removal of the material is transferred to the level-sets which represent the materials. The Boolean operation to subtract the volume penetrated by level-set ϕ_B from level-set ϕ_A is

$$\phi'_A(\boldsymbol{x}) = \min\{\phi_A(\boldsymbol{x}), -\phi_B(\boldsymbol{x})\} \ . \tag{3.2}$$

A significant advantage of this "*top layer*" approach is that material layers can be represented with sub-grid resolution. This is possible if the level-sets representing the materials are chosen to not map directly to the material regions, but are constructed "additively". An example of a three materials setup (illustrated in Figure 3.5a) is used for demonstration: A thick bottom region (blue) is covered by a thin layer (green) which is exposed only in a circular region due to a mask (red). The level-set resolution is chosen such that only a single level-set grid point falls within the vertical span of the green layer.

Figure 3.5b shows a cross section of the (narrow band) level-set grid of the domain. The colored lines mark the extracted zero-level-sets of the regions if the level-sets directly represent the material regions. Figure 3.5c illustrates the same cross section but showing the zero-level-sets, if the level-sets are constructed additively using

$$\phi_1 = \phi_{blue} \ , \tag{3.3}$$
$$\phi_2 = \phi_{blue} \cup \phi_{green} \ , \tag{3.4}$$
$$\phi_3 = \phi_{blue} \cup \phi_{green} \cup \phi_{red} \ , \tag{3.5}$$

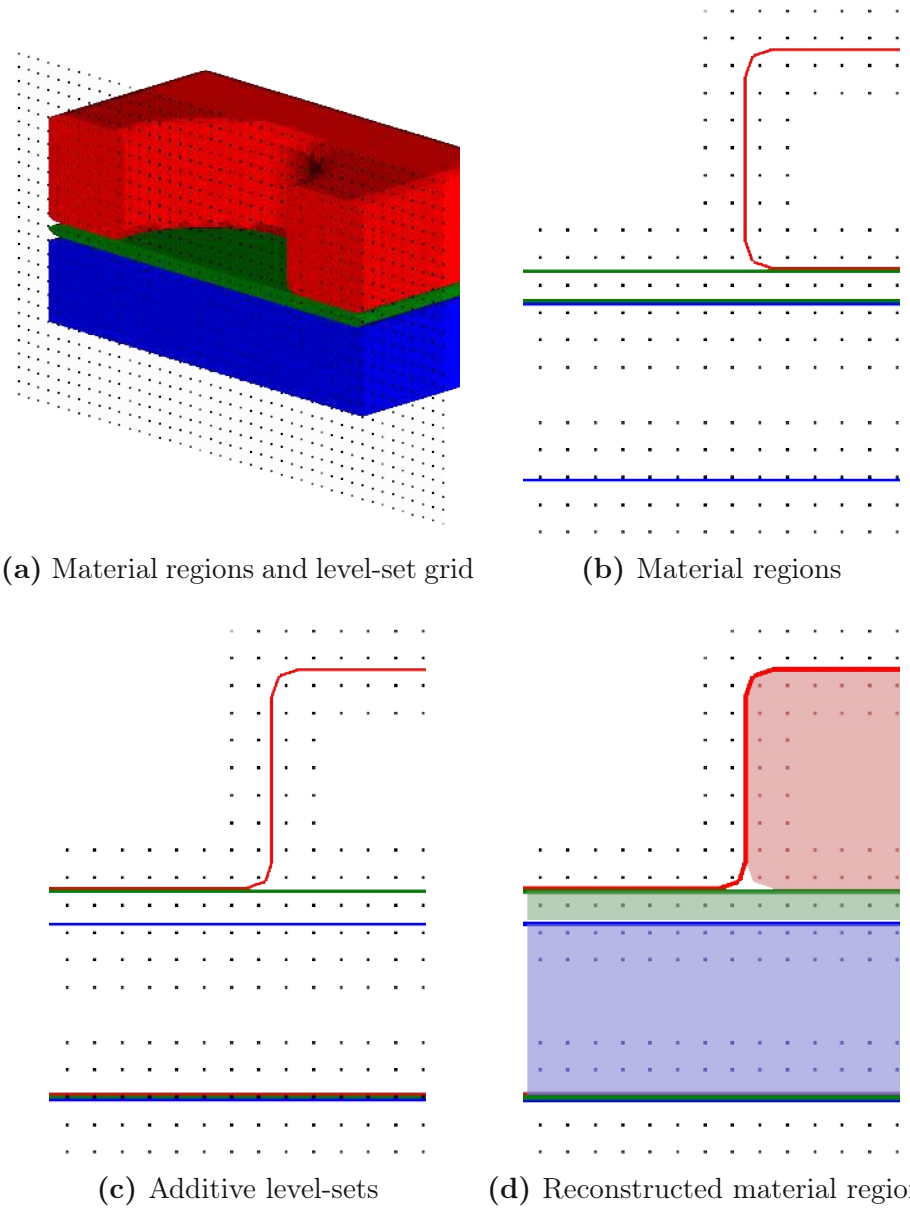
where the unions are constructed using

$$\phi_A \cup \phi_B = \min\{\phi_A(\boldsymbol{x}), \phi_B(\boldsymbol{x})\} \ . \tag{3.6}$$

Figure 3.5d once more illustrates the zero-level-sets from Figure 3.5c and additionally shows the reconstructed material regions obtained from

$$\phi_{blue} = \phi_1 \ , \tag{3.7}$$
$$\phi_{green} = \phi_2 \setminus \phi_1 \ , \tag{3.8}$$
$$\phi_{red} = (\phi_3 \setminus \phi_2) \setminus \phi_1 \ . \tag{3.9}$$

Figure 3.6 chronologically illustrates the development of the material stack using *top layer* advection and the additive scheme to represent the materials. Starting from Figure 3.6b the green layer is represented with sub-grid resolution, until it is fully etched away in Figure 3.6c. Using a straightforward scheme, the thin region of the green layer cannot be represented from Figure 3.6b onwards. It also becomes



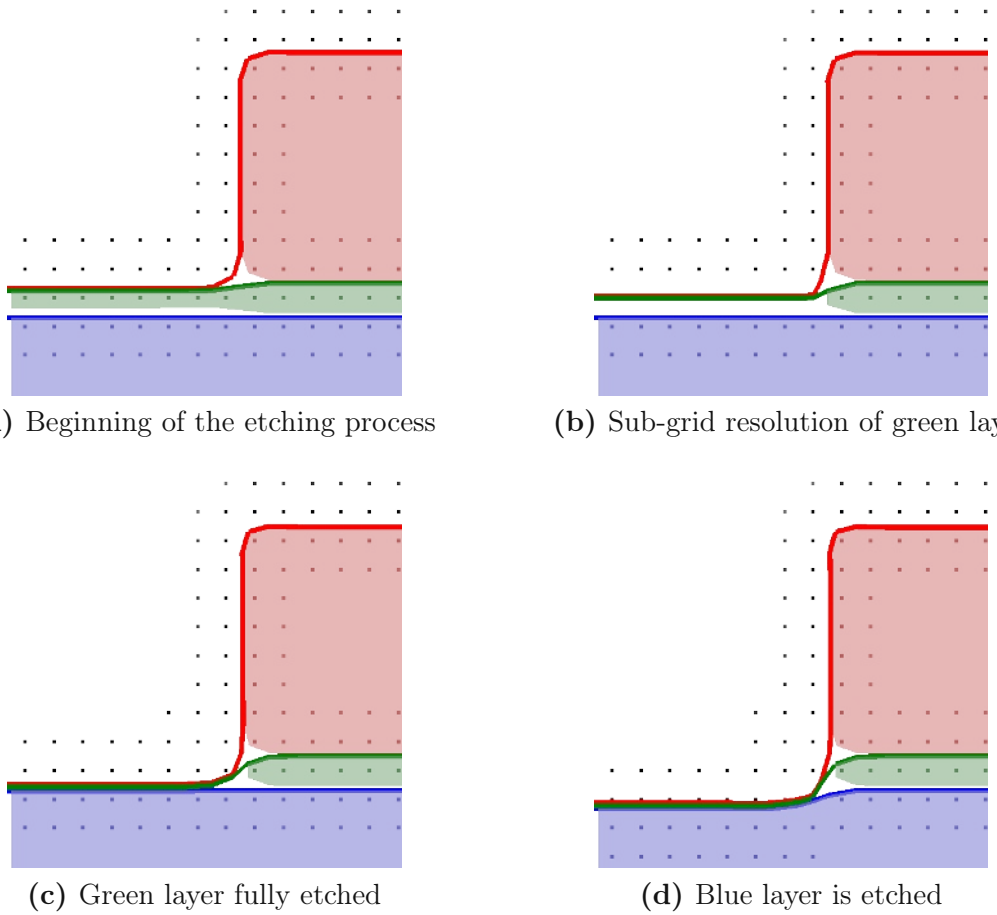
(a) Material regions and level-set grid **(b)** Material regions

(c) Additive level-sets **(d)** Reconstructed material regions

Figure 3.5: Three materials stack with thick bottom region (blue), a thin layer (green), and a mask (red). (a) Cross section through the domain and indicated level-set resolution for the cross section plane. (b) Front view on the right half of the cross section showing the narrow band level-set grid points and the extracted zero-level-sets of the material regions. (c) Representation using an additive scheme from bottom to top in order blue, green, and red. (d) Material regions reconstructed from the level-set of the additive scheme.

apparent that a void is formed between the green and the blue layer already in Figure 3.6a; this is due to the nature of the level-set function which is a scalar field and does not hold directional information.

In a time step where a region of a layer is fully etched and the underlying material becomes active (cf. Figure 3.6c), the surface velocities of the involved materials must be averaged: If only the surface velocity for the green layer is considered for the



(a) Beginning of the etching process

(b) Sub-grid resolution of green layer

(c) Green layer fully etched

(d) Blue layer is etched

Figure 3.6: Development of the material stack from Figure 3.5 during an isotropic etching simulation using the additive scheme. The zero-level-sets of the additive scheme are shown (colored lines) together with the reconstructed material regions for four different stages in chronological order: (a) The green layer representable as one horizontal layer of level-set grid points is still enclosed. (b) The green layer continues to be representable using the additive scheme but the reconstruction using Boolean operations fails to represent the green layer in the thin region. (c) The blue layer becomes the active layer where the green layer is fully etched away. (d) The blue layer is etched further.

full time step, the surface advection speed is too slow or too fast, if the surface velocity for the blue material is faster or slower, respectively. This is corrected by adopting the surface velocity in such cases and using a convex combination V_{avg} of the velocities according to the initial distance of the layers weighted with the corresponding surface speeds

$$\Delta t V_{avg} = \Delta t_g V_g + \Delta t_b V_b \ , \tag{3.10}$$

$$\Delta t = \Delta t_g + \Delta t_b \ , \tag{3.11}$$

$$\Delta t_g = \frac{\Delta x_{g \to b}}{V_g} \ , \tag{3.12}$$

where $\Delta x_{g \to b}$ is the initial distance between the green (g) and the blue (b) layer.

3.1.3 Velocity Extension

The velocity extension is implemented in the most straightforward approach [45]: For each level-set cell in the narrow band, the velocity of the closest point on the surface is identified and used. A step along the gradient weighted with the negated level-set value is performed and the search for the nearest velocity value is started at the resulting position (which is close to the surface if the level-set has a signed-distance property). The nearest neighbor search is accelerated by using *OpenVDB*'s space-partitioning acceleration structure `PointIndexGrid`. The velocity extension in the *ghost regions* is handled equally but with a preceding mapping of the query location into the domain according to the boundary conditions.

3.1.4 Ray-Surface Intersection

The ray-surface intersection queries (which are extensively used during the calculation of the particle transport) must consider the confined domain with prescribed boundary conditions:

- If a reflective boundary of the domain is intersected, the direction is reflected and the origin is set to the intersection point.
- If a periodic boundary condition is intersected, the direction is maintained but the origin is translated to the opposing domain boundary.

A maximum number of domain boundary intersections is defined to prevent infinite domain intersections for ray directions which are (nearly) horizontal and do not intersect with the geometry.

3.2 Software Design

The simulation platform is not intended to be an operational process simulator but is developed to support the evaluation of novel approaches for the particle transport. To that end, from a design perspective, the requirements are

- to support multiple implementations of the same computational task,
- to decouple individual computational tasks,
- to provide explicit interfaces between the computational tasks, and
- to not constrain new implementations of computational tasks by choosing a loose coupling between the interfaces.

A loose coupling of the individual computational tasks is achieved by the use of abstract interface classes. The relations between the interfaces are not enforced by the design: This allows to integrate new implementations of computational tasks without design constraints but provides guidance to reduce redundancy once an implementation matures. Figure 3.7 provides an overview of the relation between

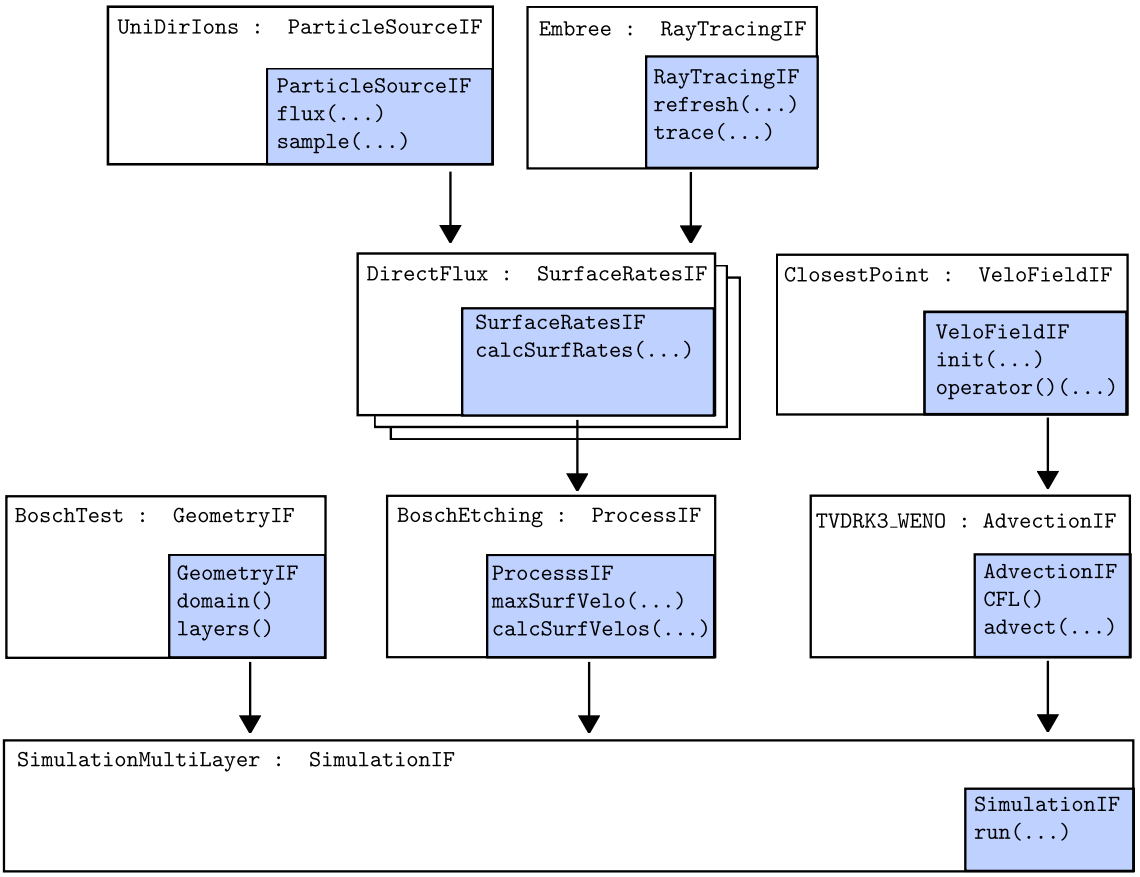


Figure 3.7: Main interface classes (blue boxes, postfix `IF`) and examples for corresponding implementations.

the interface classes and the names of the implementations of the interfaces used for the test case in Section 3.3.3. To support mixed-precision approaches, the floating point data types are templated using C++ template metaprogramming, i.e., different floating points types can be configured for each task. The floating point templatization and some details of the interfaces are not reflected in the following to keep the listings concise.

The outermost level is the `SimulationIF` class (cf. Figure 3.7). The only interface method is a function to trigger the simulation of a process for $T =$ `simtime`. The class shown in Listing 3.1 sketches a multi-material simulation implementing `SimulationIF` which utilizes the other interface classes as intended by the design. A static function (`create`) prepares the arguments for the interface constructor and returns a shared pointer object. The `run`-method first determines the maximum surface velocity of the process and calculates the maximum time step according to the advection scheme. Then, a point cloud of surface velocities is calculated using the `calcSurfVelos`-method of `ProcessIF`. The *top layer* is then advected according to these surface velocities using the `advect`-method of `AdvectIF`. Finally, the advection of the top layer is transferred to the other material layers. This sequence is repeated until `t` == `simtime`.

The `GeometryIF` class provides access to the domain (e.g., extents and boundary

Listing 3.1: Process Simulation Implementation

```
class SimulationMultiLayer : public SimulationIF {
private:
  GeometryIF::Ptr mGeometry;
  ProcessIF::Ptr mProcess;
  AdvectIF::Ptr mAdvect;
public:
  using Ptr = shared_ptr<SimulationMultiLayer>;
  static SimulationMultiLayer::Ptr Create([...]) {
    // [set/create members]
    return make_shared<SimulationMultiLayer>([...]);
  }
  SimulationMultiLayer([...]) : mGeometry([...]), [...] {}
  void run(double simtime) {
    double t = 0;
    while (t < simtime) {
      double vmax = mProcess->maxSurfVelo();
      double dtmax =
          mAdvect->CFL() * mGeometry->Domain.dx / abs(vmax);
      double dt = (t + dtmax) < simtime ? dtmax : simtime - t;
      vector<Vec3f> points;
      vector<double> velos;
      mProcess->calcSurfVelos(mGeometry, dt, points, velos);
      mAdvect->advect(t, dt, mGeometry->domain(), points, velos,
                      mGeometry->layers().back());
      // [boolean operations with other material layers]
      t = t + dt;
    }
  }
};
```

conditions) and the material layers. It is also used for the creation of the initial geometry (using CSG operations between level-sets) and saving of intermediate and final results.

A process is implemented using the interface shown in Listing 3.2. The first method (`maxSurfVelo`) calculates the maximum possible surface speed of the process. The second method's task is to create a set of points on the surface (`points`) and calculate the corresponding surface velocities (`velos`); it has read access to the geometry, i.e., the domain and the material layers. The time step `dt` is provided to allow for the averaging of velocities as discussed in Section 3.1.2.

The surface rates for a particle source are interfaced using `SurfaceRatesIF` (Listing 3.3). An implementation of this class uses an implementation of `RayTracingIF` to conduct the ray-surface intersection tests. An interface for a particle source (`ParticleSourceIF`) is used to access the properties of the source or to generate random samples according to the emission properties of the particle source. The `maxSurfRate`-method returns the maximum possible surface rate. The `calcSurfRates`-method calculates the surface rates at all triangles (`ratesAtTriangles`) of the surface (`mesh`), i.e., the extracted surface of the zero-level-set of the top layer; `mesh` holds additional information for each triangle, e.g., which material is active

Listing 3.2: Process Interface

```
class ProcessIF {
public:
  using Ptr = shared_ptr<ProcessIF>;
  virtual double maxSurfVelo() const = 0;
  virtual bool calcSurfVelo(const double dt,
                            const GeometryIF::Ptr geometry,
                            vector<Vec3f> &points,
                            vector<double> &velos) = 0;
};
```

or a quality measure (as the extraction algorithms potentially generate low quality triangles).

Listing 3.3: Surface Rates Interface

```
class SurfaceRatesIF {
public:
  using Ptr = shared_ptr<SurfaceRatesIF>;
  virtual double maxSurfRate() = 0;
  virtual void calcSurfRates(const GeometryIF::Ptr geometry,
                             const Mesh &mesh,
                             vector<double> &ratesAtTriangles) = 0;
};
```

A source of particles is implemented according to `ParticleSourceIF` (Listing 3.4). The interface methods aim at two different approaches for the particle transport calculation:

`flux(...)` provides the emitted flux into direction `dir` and is used for *bottom-up* particle transport calculations.

`sample(...)` generates a random emission direction `dir` together with a scalar weight (the return value) and is used for *top-down* particle transport calculations. The same approach as in [5] is chosen to generate the random directions.

Listing 3.4: Particle Source Interface

```
class ParticleSourceIF {
public:
  using Ptr = shared_ptr<ParticleSourceIF>;
  // flux emitted into direction
  virtual double flux(const Vec3f &dir) const = 0;
  // create random direction which resembles the distribution
  virtual double sample(Vec3f &dir) const = 0;
};
```

The ray tracing interface `RayTracingIF` (Listing 3.5) provides ray-surface intersection queries via a `trace`-method. The `trace`-method traces a ray from origin `org`

into direction `dir` through the domain. If a domain boundary is intersected, the ray's origin and direction are updated accordingly and tracing proceeds. The method reports the distance of the closest intersection with the surface `hitDistance`, the normal direction of the surface at the intersection location `hitNormal`, the ID of the intersected primitive `primitiveID`, and the number of performed ray casts `numCasts` (i.e., number of boundary intersections). The return value signals an intersection with the surface (`true`) or no intersection (`false`). The `refresh`-method is used

Listing 3.5: Ray Tracing Interface

```
class RayTracingIF {
public:
  using Ptr = shared_ptr<RayTracingIF>;
  // refresh the scene
  virtual void refresh(const Domain &domain,
                       const vector<Vec3f> &points,
                       const vector<Vec3i> &triangles) = 0;
  // trace ray
  virtual bool trace(const Vec3f &org, const Vec3f &dir,
                     double &hitDistance, Vec3f &hitNormal,
                     int &primitiveID, int &numCasts) const = 0;
};
```

to refresh the scene, i.e., to re-build the acceleration structure according to the new geometry given by a triangle mesh (`triangles` and `points`).

An interface intended for the use with implicit ray tracing (i.e., the signed-distance field is used directly for ray casting) is implemented analogously; obviously, it lacks the information of an intersected `primitiveID` but provides an intersection location.

The advection of the *top layer* level-set is interfaced via `AdvectIF` (Listing 3.6). Only one method is defined which advects a level-set inside the domain from time `t` to `t + dt` using the surface velocities (`velos`) provided at the corresponding locations on the surface (`points`). The advection of the surface includes normalization of the level-set and an update of the narrow band, i.e., the narrow band has to be moved as the zero-level-set evolves.

Listing 3.6: Level-Set Advection Interface

```
class AdvectIF {
public:
  using Ptr = shared_ptr<AdvectIF>;
  // advect level-sets according to surface velocities
  virtual void advect(const double t, const double dt,
                      const Domain &domain,
                      const vector<Vec3f> &points,
                      const vector<double> &velos,
                      LevelSet::Pt > &levelset) = 0;
};
```

The extension of the surface velocities into the narrow band is provided by `VeloFieldIF` shown in Listing 3.7. The `refresh`-method is intended to prepare (or conduct) the extension into the narrow band. The extended velocities are accessed via the overloaded `operator()`.

Listing 3.7: Velocity Field Interface

```
class VeloFieldIF {
public:
  using Ptr = shared_ptr<VeloFieldIF>;
  virtual void refresh(const Domain &domain,
                       const LevelSet::Ptr levelset,
                       const vector<Vec3f> &points,
                       const vector<double> &velos ) = 0;
  virtual double operator()(const Coord &ijk) const = 0;
};
```

The relation of the interfaces introduced above to the sequence of computational tasks throughout a process simulation is indicated in Figure 3.1. The tasks in the left and right column are handled by an implementation of `GeometryIF`. The tasks in the center column are handled in the `run`-method of `SimulationIF`: First, `GeometryIF` is used to extract the *top layer* surface. Then, `ProcessIF` is used to calculate the surface rates and surface velocities (utilizing `ParticleSourceIF`, `RayTracingIF`, and `SurfaceRatesIF`). It follows the velocity extension (`VeloFieldIF`) and the advection (`AdvectIF`).

The test cases in the following sections use computationally inexpensive implementations of `ProcessIF`, i.e., a calculation of the particle transport is not required for the surface velocities. The runtime of the simulations are therefore dominated by the advection of the level-sets and extraction of the surface representation.

3.3 Test Cases

The multi-material surface advection capabilities of the platform are validated using three test cases. Simple surface velocity models are used which are independent of the particle transport. Therefore the surface advection and maintenance of the multi-material level-sets becomes the main computational load. Different spatial resolutions are used, the final profiles are compared, and the performance is tracked throughout the simulations. The main computational tasks for the following test cases is the advection of the surface including velocity extension and re-normalization.

3.3.1 Enright Test

The *Enright test* is described in [69] and is a standard test for benchmarking level-set data structures and related numerical methods [62][70]. The test is straightforward to setup: A sphere with radius 0.15 is placed with its center at $\boldsymbol{x} = [0.35, 0.35, 0.35]$.

The simulation domain is the unit box: $[0,0,0]$ to $[1,1,1]$ (cf. Figure 3.8). The velocity field $\boldsymbol{V}(\boldsymbol{x},t)$ is a three-dimensional incompressible flow field modulated with a period $T=3$, i.e., a multiplication with $\cos(t\frac{\pi}{3})$, which results in

$$\boldsymbol{V}(\boldsymbol{x},t) = \begin{bmatrix} \cos(t\frac{\pi}{3})[2\sin^2(2\pi x)\sin(2xy)\sin(2\pi z)] \\ \cos(t\frac{\pi}{3})[-\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)] \\ \cos(t\frac{\pi}{3})[-\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)] \end{bmatrix} . \tag{3.13}$$

The sphere is mangled through this velocity field resulting in a very thin structure at $T=1.5$ where the velocity field and thus the deformation as well is reversed (due to the temporal cosine modulation). The analytical result is again the original structure (sphere) for $T=3.0$.

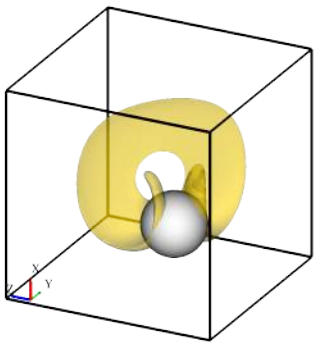


Figure 3.8: Initial position of the sphere for the *Enright test* in the unit domain (gray) and deformed sphere at $T=1.5$ (yellow).

In Appendix A.1, Figure A.1-A.4 illustrate the deformed states of the sphere from $T=0.0$ to $T=3.0$ for various resolutions. For the lowest resolution (128^3, Figure A.4) the thin regions are not representable from $T=1.2$ onwards (cf. Section 3.1.2), leading to a drastic loss in volume at the final time $T=3.0$. When the resolution in is increased (cf. Figure A.1-Figure A.3), the loss in volume is decreased.

Table 3.1 shows the results using the *OpenVDB* data structure to store the level-set and performing the surface advection using a TVD-RK3 time integration scheme and WENO schemes for spatial discretization; three iterations (i.e., time steps) of a flow-based normalization are performed after the advection using the same WENO schemes and a TVD-RK1 time integration.

3.3.2 Material Dependent Isotropic Etching

The test configuration (*isotropic test*) consists of four stacked layers of different materials M. The surface velocity model is an isotropic etch rate:

$$V_n(\mathbf{x}) = f(M(\mathbf{x})) = \begin{cases} 0.07, & \text{if } M(\mathbf{x}) = M_4 \\ 1, & \text{if } M(\mathbf{x}) = M_3 \\ 0.05, & \text{if } M(\mathbf{x}) = M_2 \\ 1, & \text{if } M(\mathbf{x}) = M_1 \end{cases} \tag{3.14}$$

Domain resolution	Memory [MB]	Advection [s]	Performance [MAV/s]	Active voxels [million]
128^3	0.6/1.0	0.01/0.03	2.3/2.8	0.027/0.078
256^3	1.6/4.3	0.04/0.13	2.7/3.0	0.11/0.39
512^3	5.4/18	0.15/0.56	2.9/3.1	0.44/1.7
1024^3	20/76	0.57/2.4	2.9/3.1	1.7/7.2
2048^3	80/330	2.4/10	2.9/3.1	7.1/29

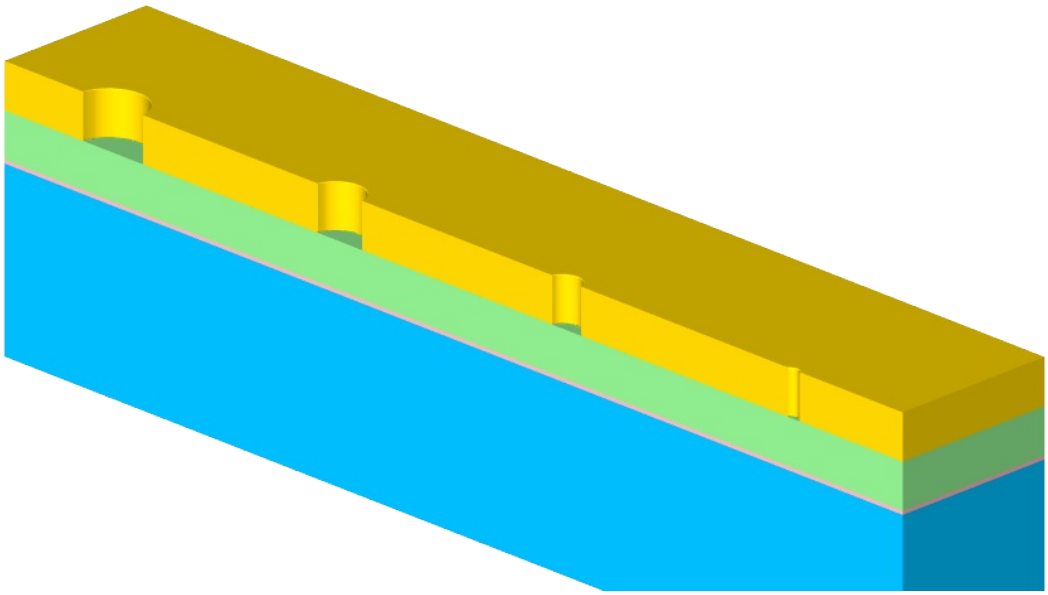
Table 3.1: Performance characteristics for the *Enright test* for resolutions from 128^3 to 2048^3 (min/max) test on WS1 (cf. Section 3.4). Column 1: Memory footprint of the VDB data structure (single-precision floating point representation). Column 2: Time for the advection step (TVD-RK3, WENO) including normalization (3 iterations, flow-based TVD-RK1, WENO) and tracking of the narrow band. Column 3: Advection performance in MAV/s (million active voxels per second). Column 4: Number of active voxels.

The lateral domain dimensions are 6×1 (cf. Figure 3.9). The boundary conditions are reflective in both lateral directions. The thicknesses of the layers from top to bottom are 0.25 (M_4), 0.25 (M_3), 0.02 (M_2), and 0.98 (M_1), respectively. The top layer (M_4) has four cylindrical holes of varying diameters 0.4, 0.3, 0.2, and 0.1. The centers are located on one of the longer lateral boundaries of the domain and are equidistantly spaced with a distance of 1.5.

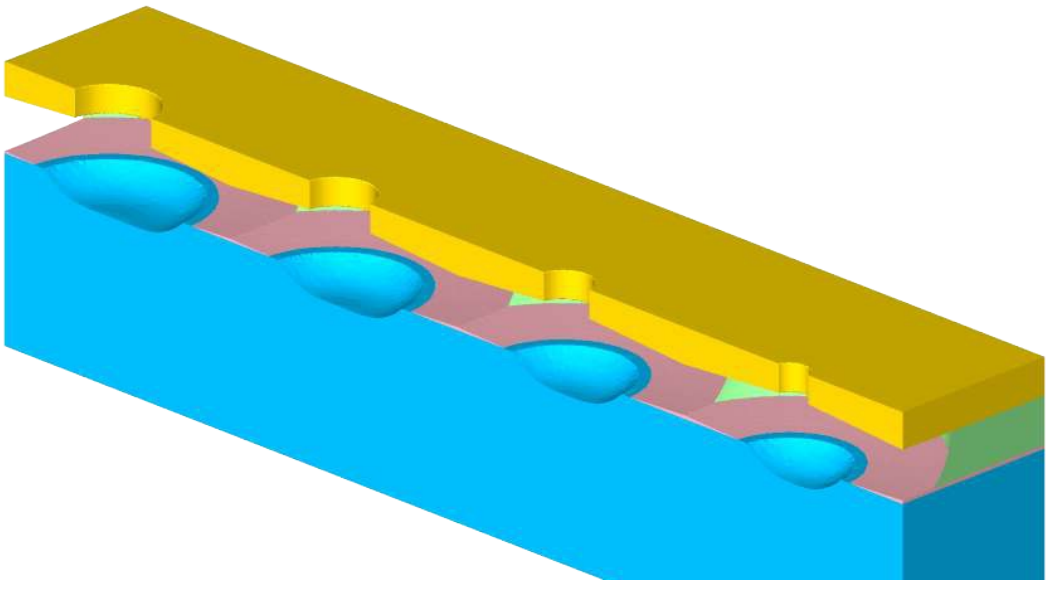
Three different level-set resolutions were used: 1/64, 1/128, and 1/256, corresponding to lateral dimensions of 1536×256, 768×128, and 384×64 grid cells for the simulation domain. The simulation is run until time $T = 1.0$ using a TVD-RK3 time integration scheme with $\alpha_{CFL} = 0.4$. The simulation results for $T = [0, 1]$ are illustrated in Figures A.5-A.7. The thin material layer (M_2) has a low etch rate compared to the materials above and below; for the lowest resolution ($1/64 = 0.0156$) the layer is just a bit thicker than one grid cell. Nevertheless, the multi-material level-set representation described above allows to represent this thin layer with sub-grid thickness (cf. Section 3.1.2) during the level-set advection.

If the material regions are reconstructed from the multi-material level-sets (using Boolean operations), the thin layer vanishes in regions where no grid cell center is situated between the wrapping level-sets. This effect is visualized in Figure 3.10 where this sub-grid resolution is shown for the thin layer of M_2 near the crater in M_1 at $T = 0.7$.

Figure 3.11 shows the cross section for the hole of diameter 0.2 at different simulation times for all three resolutions. The low etch rate of M_2 slows etching in the downward direction, until the thin layer is first opened at the center line of the hole.



(a) Initial material regions



(b) Material regions at $T = 0.75$

Figure 3.9: (a) Initial material regions M_4(yellow), M_3(green), M_2(red), and M_1(blue). (b) Material regions for $T = 0.75$.

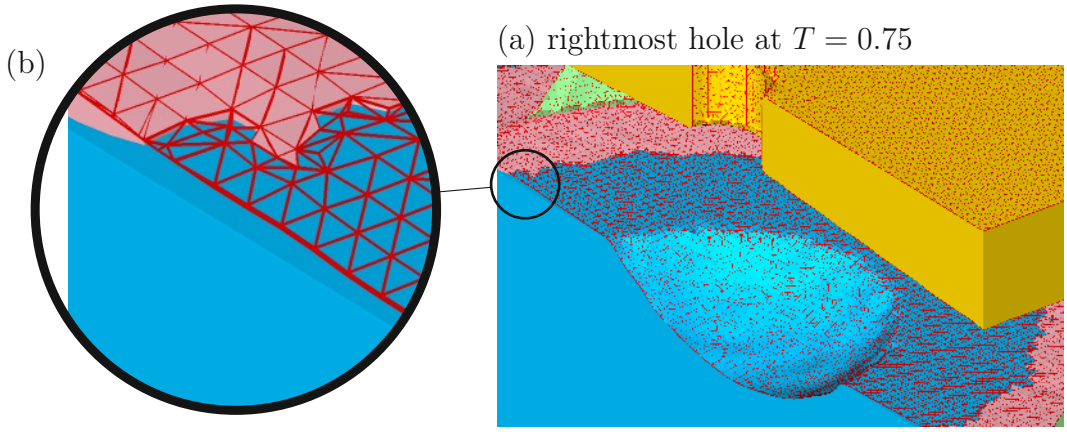


Figure 3.10: Material regions at $T = 0.75$ for the rightmost hole. (a) The explicit representation of the *top layer* level-set is shown (red triangle mesh). (b) A spot near the crater in material M_1 is magnified to illustrate the sub-grid resolution of the thin material later (M_2). If no grid cell is between the two wrapping level-sets, the thin layer vanished during a Boolean operation between the wrapping level-sets. This is visible in the horizontal region near the crater. Nevertheless, the layer is still represented through the wrapping level-sets and considered during the surface advection.

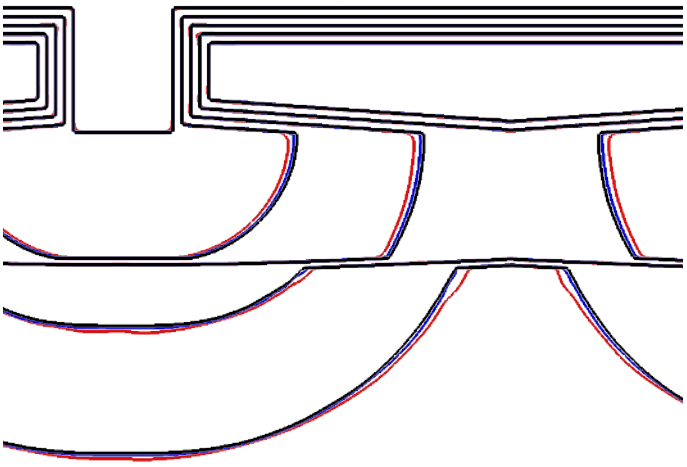


Figure 3.11: Cross section of the *top layer* level-set for resolutions 1/256(black), 1/128(blue), and 1/64(red). For time $T = 0.0, 0.25, 0.5, 0.75$, and 1.0, the *top layer* level-set for all three resolutions is shown.

3.3.3 Simple Bosch Process

The Bosch process is a technique to fabricate high aspect ratio structures which are obtained by alternating a passivation step with an etching step. The passivation layer protects the sidewalls in the following directional etching step. In the etching step, accelerated ions remove the passivation layer primarily on horizontal surface regions, i.e., at the open top surface and the bottom of the structure. Once the passivation layer is removed at the bottom, the substrate is also etched isotropically by a neutral etchant species.

For this test case (*Bosch test*), etching and deposition are assumed to be isotropic and the accelerated ions are modeled to have perfect vertical trajectories. No re-emissions are considered allowing to simplify modeling the ion flux by a single vertical (upward) visibility check weighted with the projected surface area. This simple model is deliberately chosen as this test case is solely conducted to evaluate the capabilities of the framework.

The involved materials are a mask M_{mask} patterned with the desired structure, a substrate M_{subs} into which the pattern is transferred, and a passivation material M_{pass}. This results in an isotropic surface velocity for the deposition of the passivation layer M_{pass}

$$V_n(\mathbf{x}) = f(M(\mathbf{x})) = \begin{cases} 0.00025, & \text{if } M(\mathbf{x}) = M_{pass} \end{cases}, \tag{3.15}$$

and a combined (isotropic and directed) surface velocity for the etching step

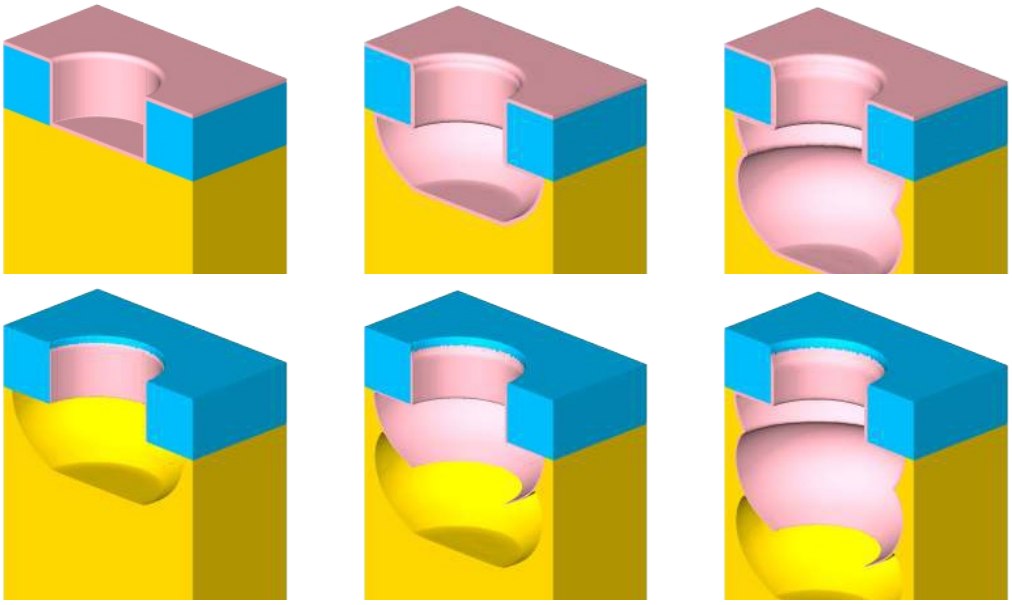
$$V_n(\mathbf{x}) = f(M(\mathbf{x}), \boldsymbol{n}(\boldsymbol{x})) = \begin{cases} 0.000875 + 0.00275(\boldsymbol{n} \cdot \boldsymbol{\omega}_{up}), & \text{if } M(\mathbf{x}) = M_{pass} \\ 0.00025 + 0.0005(\boldsymbol{n} \cdot \boldsymbol{\omega}_{up}), & \text{if } M(\mathbf{x}) = M_{mask} \\ 0.003 + 0.00295(\boldsymbol{n} \cdot \boldsymbol{\omega}_{up}), & \text{if } M(\mathbf{x}) = M_{subs} \end{cases}, \tag{3.16}$$

where $\boldsymbol{n}$ is the surface normal direction and $\boldsymbol{\omega}_{up}$ is an upward unit vector. The second summand is only considered, if the upward visibility check (a single upward ray is traced) evaluates to `false`, i.e., the source is visible. The duration of the passivation step is $T = 5$ and the duration of the etching step is $T = 12$. The simulation of 20 cycles is conducted using a domain with lateral dimensions 1×0.5 and reflective boundary conditions. A mask (M_{mask}) of thickness 0.25 is patterned with a cylindrical hole of radius 0.25 with its center midway on one of the longer domain boundaries.

Figure 3.12 illustrates the results of the first 5 processing steps.

The results for all 20 cycles are provided in Figures A.8-A.11 in Appendix A.3 for resolutions 1/256, 1/128, 1/64, and 1/32. It becomes apparent that the sub-grid resolution of thin layers is essential especially at low resolutions, as the thin passivation layer would not be representable otherwise.

Figure 3.13 compares the final results for four different resolutions. Larger differences in the absolute position of the side wall are noticeable for the lower resolutions.



(a) First cycle **(b)** Second cycle **(c)** Third cycle

Figure 3.12: Material regions of the *Bosch test* for resolution 1/256. (a)-(c): Material regions after each of the first 3 cylces, i.e., alternating deposition (top) and etching (bottom). The corresponding final material regions after 20 cycles are shown in Figure A.8f in Appendix A.3.

(a) Top **(b)** Middle **(c)** Bottom

Figure 3.13: Final cross sections for resolutions 1/256(black), 1/128(blue), 1/64(red), and 1/32(green). All three material layers (e.g., cf. Figure A.8g) are shown at (a) the top, (b) the middle, and (c) at the bottom of the hole.

3.4 Benchmarks

The individual computational tasks of the simulation platform were benchmarked on two hardware configurations:

Workstation 1 (WS1). Intel Core-i7-4790K system, 4 physical cores (8 logical cores), 8 MB of shared L3 cache, 32 GB of main memory.

Workstation 2 (WS2)[2]. Dual CPU Intel Xeon E5-2650v2 system, each CPU has 8 physical cores (16 logical cores), 20 MB of L3 Cache, 64GB of main memory.

Utilizing all available cores (including hyper-threading), the *isotropic test* is benchmarked on WS1: The runtime of the main computational tasks (resolution 1/256) is shown in Figure 3.14. The majority of runtime is utilized by the surface advection. The second largest runtime is used for the surface representation, which includes the extraction of an explicit surface of the *top layer* level-set and the detection of the active layer for each triangle of the *top layer*. The runtime per time step doubles throughout the simulation. This increase is due to the increasing surface area; from $T = 0.9$ onwards, the surface area and consequently the runtime decreases.

Figure 3.15 shows the runtime of the main computational tasks throughout the simulation of the *Bosch test*: Also here, the dominating runtime during the deposition steps is the surface advection. During the etching steps the surface representation and the surface rates increase: The active material region must be determined leading to a larger runtime for the surface representation, and the upward visibility test effects the runtime for the surface rates.

To evaluate the parallel scalability of the surface advection, the first time steps of the *Enright test* for resolution 2048^3 (7.1 MAV), the *isotropic test* for resolution 1536×256 (7.0 MAV), and the *Bosch test* for resolution 256×128 (0.5 MAV) were benchmarked on a 16-core WS2. Figure 3.16 illustrates the results for up to 32 threads: The speedups are 12, 10, and 7.5 for 16 threads, respectively.

[2]Represents a single node on the Vienna Scientific Cluster 3 (VSC-3); `https://vsc.ac.at`

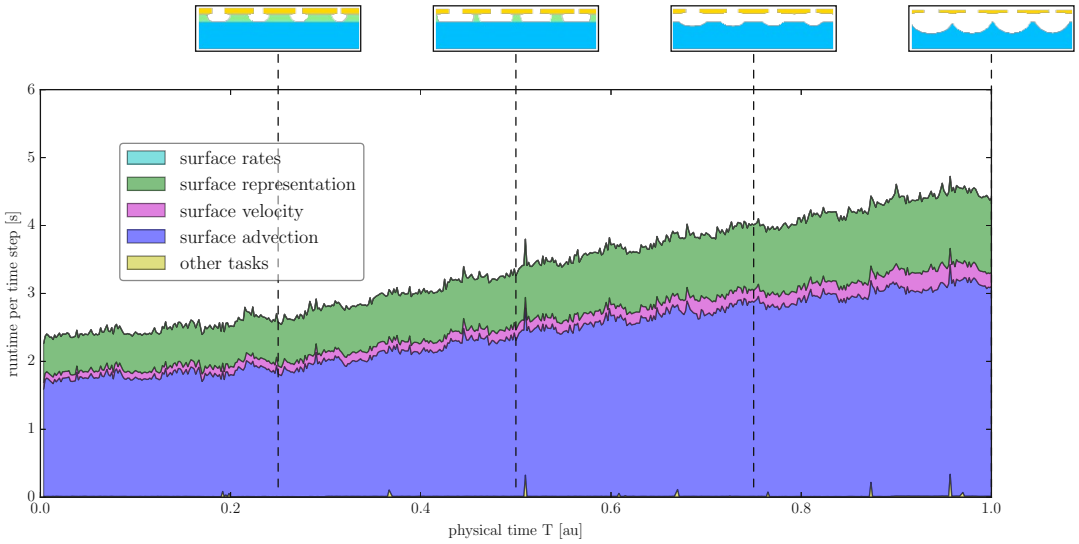


Figure 3.14: Runtime of the main computational tasks for all time steps ($T = 0$ to $T = 1.0$) of the *isotropic test* illustrated in Figure A.5. The lateral resolution of the domain is 1536×256. At the top, the corresponding cross sections of the domain are shown for times $T = 0.25, 0.5, 0.75$, and 1.0. The simulation was performed on WS1.

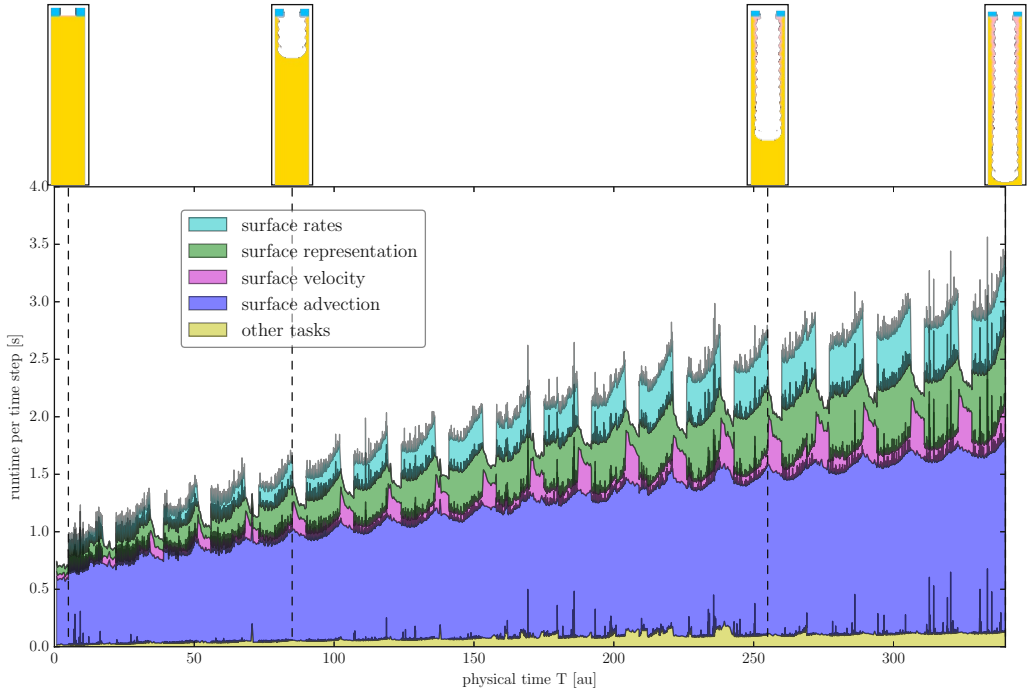


Figure 3.15: Runtime of the main computational tasks for all time steps ($T = 0$ to $T = 340$) of the *Bosch test* illustrated in Figure A.8. The lateral resolution of the domain is 256×128. At the top, the corresponding cross sections of the domain are shown after cycle 1 ($T = 17$), cycle 5 ($T = 85$), cycle 10 ($T = 170$), and cycle 20 ($T = 340$). The simulation was performed on WS1.

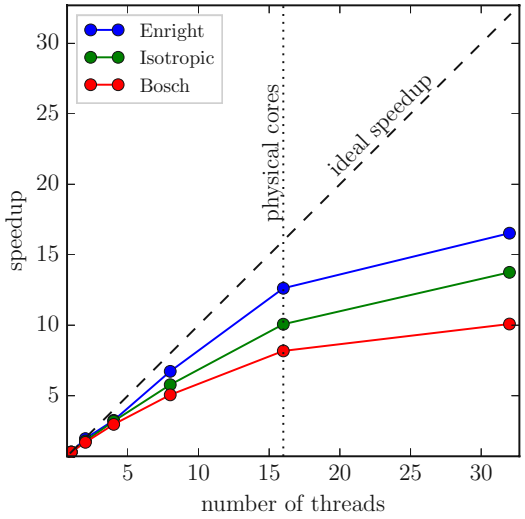


Figure 3.16: Parallel scalability of the surface advection for the *Enright test*, the *Bosch test*, and the material dependent isotropic etching test case. The benchmarks were performed on WS2.

3.5 Summary

The developed simulation platform provides important features to validate new approaches for the particle transport calculation: Multi-material advection, sub-grid resolution of thin layers, periodic and reflective boundary conditions, and an interface for efficient ray-surface intersections using external libraries. These features allow to qualify new approaches for the particle transport using realistic simulation setups. The surface advection shows appropriate speedup on shared-memory systems, although the scalability of the surface advection is not primarily important (as the focus is on the performance of the particle transport), it is advantageously as a realistic distribution of runtime between the individual computational tasks is obtained.

The platform is used to implement and validate the numerical approaches in the following Chapters 4, 5, and 6, respectively.

Chapter 4

Adaptive Visibility Sampling

If the direct flux originating from one or more remote sources above the structure is required to evaluate the surface velocity model, typically either a *top-down* or a *bottom-up* approach is used to obtain the flux rates for all points on the surface. The *bottom-up* approach integrates the flux contribution from the source towards a surface point by

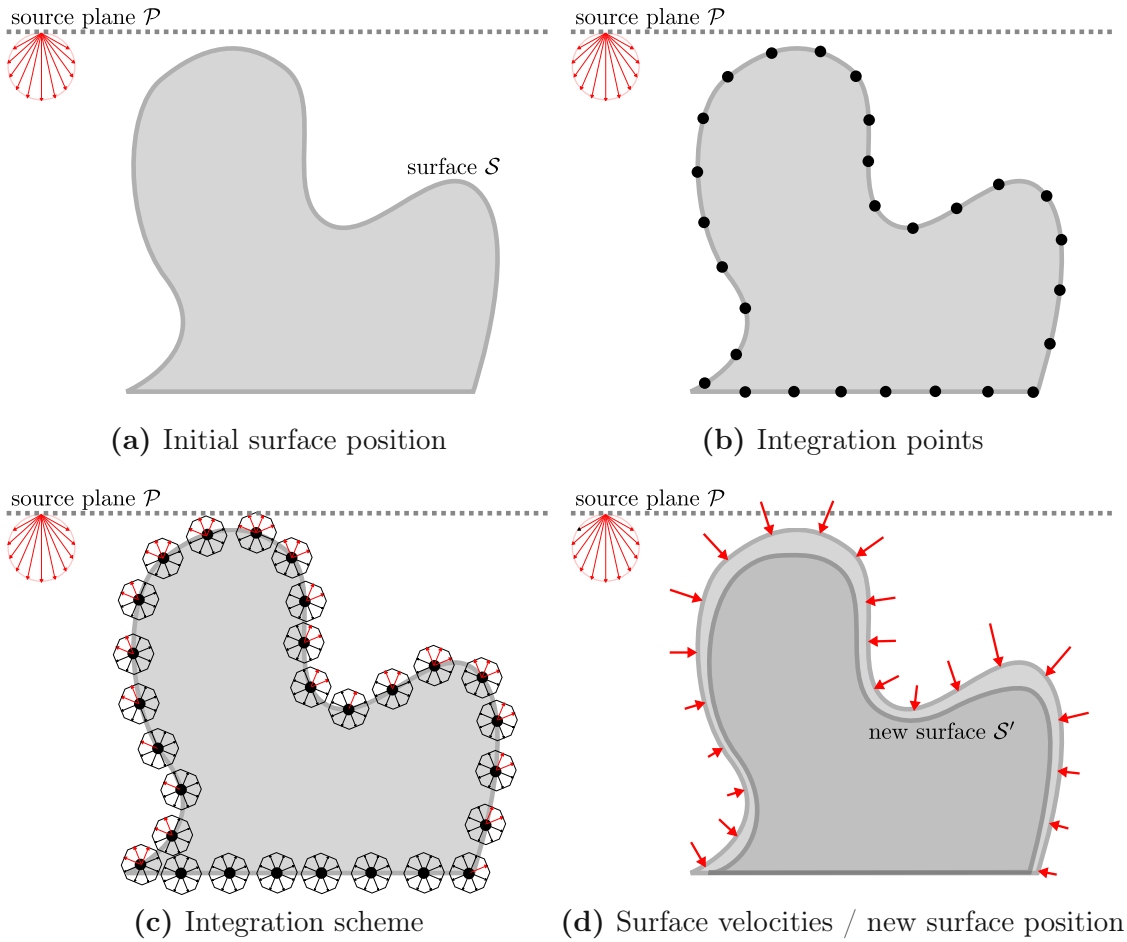
- finding all directions that are not obstructed by the geometry, and
- integrating the source contribution over these directions,

which yields the surface rates (cf. Section 2.2.1.2). The accurate calculation of these surface rates is a significant computational bottleneck in three-dimensional process simulations, especially for high aspect ratios which are increasingly required as device structures move towards full three-dimensional designs, e.g., NAND flash cells [71].

Figure 4.1 conceptually illustrates different stages in one time step of a simulation using a *bottom-up* integration scheme: (a) A set of integration points is defined on the initial surface. (b) For each integration point, a scheme defines the directions which are tested for direct visibility of the source plane. (c) The surface is advected according to the surface velocity model that requires the results of the integrations (i.e., surface rates).

For an arbitrary geometry, visibility information of the full upper hemisphere (i.e., all upward pointing vertical directions) is required; the lower hemisphere can be neglected as the source plane is always strictly above the surface. The number of required visibility tests depends on the scheme used to define the sampling directions.

The problem to discretize and access data distributed on a sphere is common in many fields [72][73][74][75]. The most straightforward (hemi)spherical tiling is to use a regular grid with respect to the two angular coordinates of a spherical coordinate system. A disadvantage is that the area of the grid cells decreases towards the poles (i.e., where the polar angle approaches $\pm\pi/2$), leading to a non-uniform sampling with respect to the subtended solid angles. Alternative sampling schemes can be derived from subdividing a spherical polyhedron. The platonic solids are a common



(a) Initial surface position **(b)** Integration points

(c) Integration scheme **(d)** Surface velocities / new surface position

Figure 4.1: Two-dimensional illustration of a surface advection using a *bottom-up* integration scheme for direct flux: (a) Initial position of the surface and source plane. (b) Integration points distributed on the initial surface. (c) Integration directions for each integration point. The red directions indicate where the source plane is directly visible. (d) Resulting surface velocities and surface position after the advection.

choice as basis for such tiling schemes [76][77][78]. The concentration of grid cells around the poles is overcome at the price of irregular coordinates.

In the following, an adaptive sampling scheme to accelerate the direct flux integration during a process simulation is presented [79][80]. It is applicable when a *bottom-up* approach (cf. Section 2.2.1.2) is chosen for the particle transport. An icosahedron, i.e., the platonic solid with the most (=20) faces, and its subdivided versions (to increase the resolution) are used to sample the spherical directions during the numerical integration of the surface rates. The presented scheme reduces the number of rays which have to be traced during the visibility calculation by only refining the sample directions where the visibility changes. The scheme is especially useful for geometric configurations as they occur in *Process TCAD*, i.e., typically the source plane is visible through one or a few larger apertures. It is shown that if the initial sampling is chosen appropriately, the accuracy of the flux integration is not reduced but the runtime is cut by 50% for higher spatial resolutions.

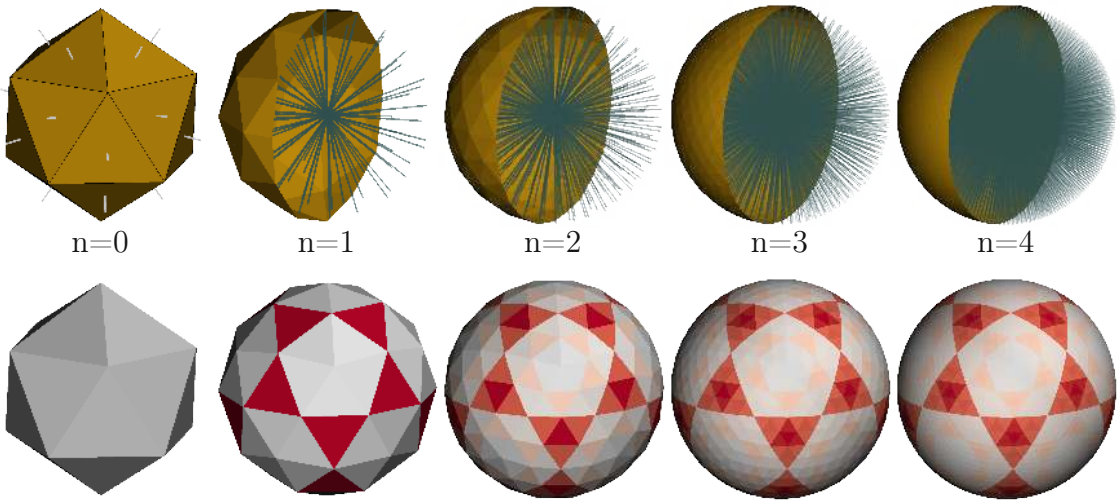


Figure 4.2: Initial icosahedron (n=0) and subdivisions up to n=4. The top row shows the integration directions originating at the center and pointing towards each centroid of the triangles of the spherical mesh. The bottom row visualizes the differences in the areas of the triangles (gray = smaller, red = larger) resulting from the subdivisions. Detailed numerical information about each subdivision step is listed in Table 4.1.

First, the properties of the hierarchically subdivided icosahedron are presented in more detail. Then, the numerical intgeration scheme of the direct flux is presented. Finally, the adaptive sampling scheme is introduced and results are shown for a cylindrical hole with aspect ratios 1 to 25.

4.1 Subdivided Icosahedron

The initial spherical tiling is provided by an icosahedron which consists of 20 equilateral triangles defined by 12 vertices on a spherical surface. To increase the resolution, the triangles are subdivided according to [81]: For each triangle,

- compute the midpoints of the edges,
- project those midpoints onto the unit sphere,
- remove the original triangle, and
- connect the three original vertices and the three new midpoints to form four new triangles.

The resulting triangles have similar shape and quality as the original triangles of the icosahedron although they differ slightly in size and are not equilateral.

Figure 4.2 visualizes an icosahedron ($n = 0$) and the first four subdivisions ($n = 1$ to $n = 4$) together with the integration directions which are defined via the centroids of the triangles. Table 4.1 provides detailed properties of the original icosahedron and the subdivision steps up to $n = 9$. The number of triangles for subdivision step

subdivisions	N_{tri}	N_{vert}	α_{res}	$\Delta_{min}/\Delta_{max}$	r_{min}
n=0	20	12	25.842	1.00000	0.9000000
n=1	80	42	12.839	0.84222	0.9750000
n=2	320	162	6.409	0.78820	0.9937500
n=3	1280	642	3.203	0.77377	0.9984375
n=4	5120	2562	1.602	0.77010	0.9996094
n=5	20480	10242	0.801	0.76918	0.9999023
n=6	81920	40962	0.400	0.76895	0.9999756
n=7	327680	163842	0.200	0.76889	0.9999939
n=8	1310720	655362	0.100	0.76888	0.9999985
n=9	5242880	2621442	0.050	0.76888	0.9999996

Table 4.1: Properties of an icosahedron and its subdivisions up to n=9: N_{tri} = number of triangles, N_{vert} = number of vertices, α_{res} = angular resolution, $\Delta_{min}/\Delta_{max}$ = ratio of triangle areas, r_{min} = smallest radius occurring in triangulation.

n is defined as

$$N_{tri}(n) = 20 \cdot 4^n. \tag{4.1}$$

The corresponding number of vertices is defined by

$$N_{vert}(n) = \frac{20 \cdot 3}{5} + \frac{20 \cdot 3}{2} \cdot \sum_{i=0}^{n} 4^i \ , \tag{4.2}$$

where the first summand originates from the initial vertices shared by five triangles; the subsequent summands originate from the subdivisions which generate vertices shared by six triangles. This difference (i.e., five or six triangles sharing a vertex) is also the reason for the subdivided triangles not being equilateral and having different areas. The angular resolution α_{res} in Table 4.1 is the angle corresponding to a spherical cap of the same area as a triangle. For subdivision step n

$$\alpha_{res}(n) = \arccos(1 - \frac{4 \cdot \pi}{N_{tri}(n)} \cdot \frac{1}{2 \cdot \pi}) \ , \tag{4.3}$$

where an equal distribution of the solid angle over all triangles is assumed. Furthermore, Table 4.1 illustrates the effect of the subdivision on the spread of triangle areas $\Delta_{min}/\Delta_{max}$.

The next section presents the numerical integration based on the subdivided icosahedron in more detail.

4.2 Numerical Integration

The integration method permits sources with an arbitrary angular distribution function $\Gamma_{src}(\mathbf{\Theta})$ with direction $\mathbf{\Theta}(\theta, \varphi)$. The integral of the direct flux

$$F_i = \int_{\Omega_{HS}} \Gamma_{src}(\mathbf{\Theta})(\mathbf{\Theta} \cdot \mathbf{n_i}) d\boldsymbol{\omega}_{\mathbf{\Theta}}, \quad \text{with} \quad d\boldsymbol{\omega}_{\mathbf{\Theta}} = sin\theta d\theta d\varphi \tag{4.4}$$

received at surface location i (surface normal = $\mathbf{n_i}$) is approximated by triangulating the hemisphere Ω_{HS} based on a subdivided icosahedron leading to

$$F_i = \sum_{j=1}^{N_{tri}} \int_{\Delta_j} \Gamma_{src}(\mathbf{\Theta})(\mathbf{\Theta} \cdot \mathbf{n_i}) dA = \sum_{j=1}^{N_{tri}} \Gamma_{src}(\mathbf{\Theta_{c_j}})(\mathbf{\Theta_{c_j}} \cdot \mathbf{n_i}) \cdot \Delta_j \ . \tag{4.5}$$

Here $\mathbf{\Theta_{c_j}}$ is the direction towards the centroid of triangle j and Δ_j is the area of triangle j. A *centroid rule* (4.6) is used in (4.5) to integrate over the area of a triangle.

$$F_{i_j} = \int_{\Delta_j} \Gamma_{src}(\mathbf{\Theta})(\mathbf{\Theta n_i}) dA = \Gamma_{src}(\mathbf{\Theta_{c_j}})(\mathbf{\Theta_{c_j}} \cdot \mathbf{n_i}) \cdot \Delta_j \tag{4.6}$$

Using a visibility function $f_{vis}(\mathbf{\Theta})$, which evaluates to 0 if a surface is intersected in direction $\mathbf{\Theta}$, and 1 otherwise, the direct flux at surface location i is

$$F_i = \sum_{j=1}^{N_{tri}} \left[f_{vis}(\mathbf{\Theta_{c_j}}) \Gamma_{src}(\mathbf{\Theta_{c_j}})(\mathbf{\Theta_{c_j}} \cdot \mathbf{n_i}) \cdot \Delta_j \right] \ , \tag{4.7}$$

where $\mathbf{n_{src}}$ is the upward pointing normal of the source plane, and $\mathbf{n_i}$ is the normal direction of the surface at position i.

Validation

A power cosine source distribution with a downward mean direction is used to validate the integration method as it provides analytical solutions for the direct flux received on horizontal and vertical surfaces. The angular distribution functions are defined as $\Gamma_{src}(\mathbf{\Theta}) = \cos(\theta)^n$. For $n = 1$, the angular distribution function is reduced to $\Gamma_{src}(\mathbf{\Theta}) = \cos(\theta)$ and therefore constitutes an ideal-diffuse source[1] (i.e., a source with a constant particle flux per solid angle and per projected source area). Figure 4.3 visualizes the normalized angular distribution of a power cosine source from $n = 1$ to $n = 1000$.

The direct flux originating from an unobstructed power cosine source on a horizontal surface is

$$F_{hori} = \int_0^{\pi/2} \int_0^{2\pi} \left[\cos(\theta)^{n+1} \right] sin\theta d\theta d\varphi = \frac{2}{n+2} \pi \tag{4.8}$$

and for a vertical surface under the same source

$$F_{vert} = \int_0^{\pi/2} \int_0^{\pi} \left[\cos(\theta)^n sin(\theta) sin(\varphi) \right] sin\theta d\theta d\varphi = 2 \int_0^{\pi/2} \left[\cos(\theta)^n sin(\theta) \right] sin\theta d\theta \ . \tag{4.9}$$

The derivations of the analytical formulations are provided in Appendix B.1.

In Figure 4.4, the numerical results are compared to the analytical solutions for sources with power cosine exponents and subdivisions up to 5. The flux rates approach the analytical solutions with a relative error below 1% for 3 subdivisions and far below 0.1% for 5 subdivisions.

[1] Ideal-diffuse sources are used in *Process TCAD* to model neutral particles.

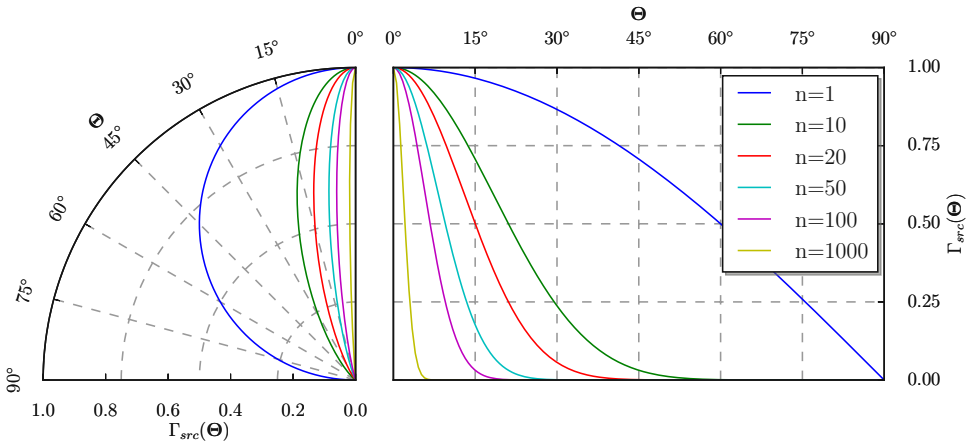
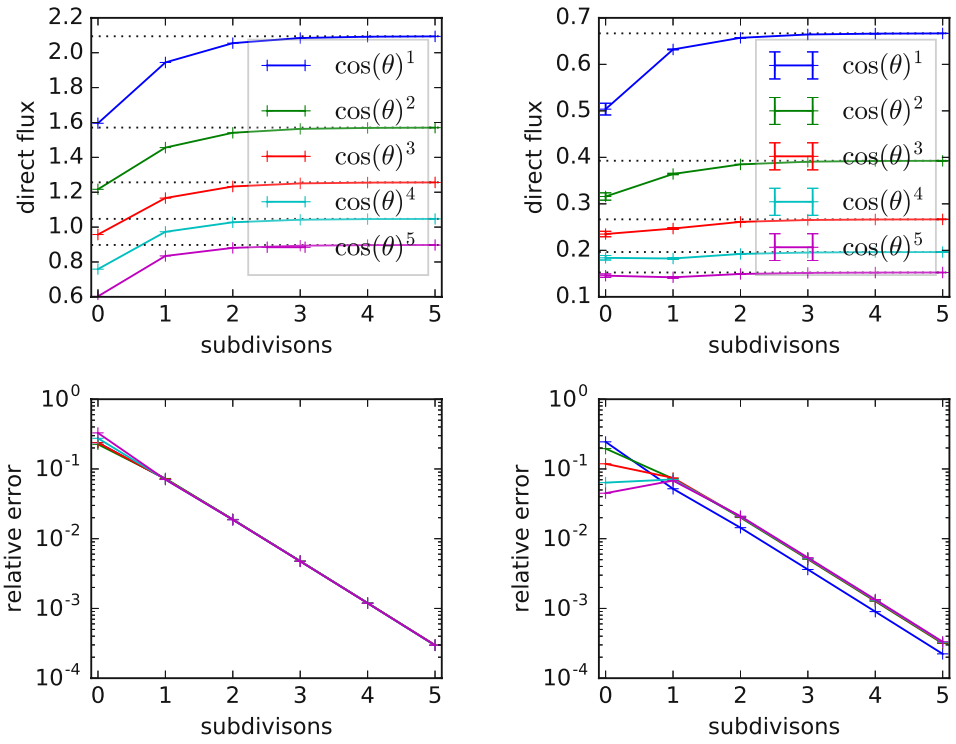


Figure 4.3: Power cosine source distribution for exponents from diffuse ($n = 1$) to very directed ($n = 1000$) in polar (left) and Cartesian (right) coordinates.



(a) Horizontal **(b)** Vertical

Figure 4.4: Direct flux received on a horizontal (a) and vertical (b) surface using the presented integration method. An unobstructed source with a power cosine distribution with exponents $n = [1, 5]$ is used. The resolution of the icosahedron is increased by using up to 5 subdivisions, resulting in a maximal angular resolution $\alpha_{res} = 0.8$ degrees (see Table 4.1). The dotted lines are the analytical solutions obtained from (4.8) and (4.9). The solid lines in (b) show the average of the flux over all four vertical faces of a cube; the error bars indicate the maximum deviation amongst the four vertical faces.

4.3 Adaptive Sampling Scheme

For each subdivision level of the icosahedron, the search directions are defined towards the centroids of the triangles of the icosahedron. The subdivision divides each triangle into four triangles on the next level, which cover the same area when projected onto the unit sphere (cf. Figure 4.2). Instead of evaluating all directions on the final subdivision level (*maxlevel*), the presented adaptive scheme starts on a lower subdivision level (*minlevel*). After evaluating all directions on *minlevel*, the boundary of the aperture is identified and all directions in the neighborhood of the boundary are re-evaluated on the next level. The aperture boundary is defined as all triangles which are connected to a vertex which shares triangles with mixed visibility information (cf. blue points in Figure 4.5). This procedure is repeated until *maxlevel* is reached. The adaptive sampling scheme is conceptually illustrated for two dimensions in Figure 4.5.

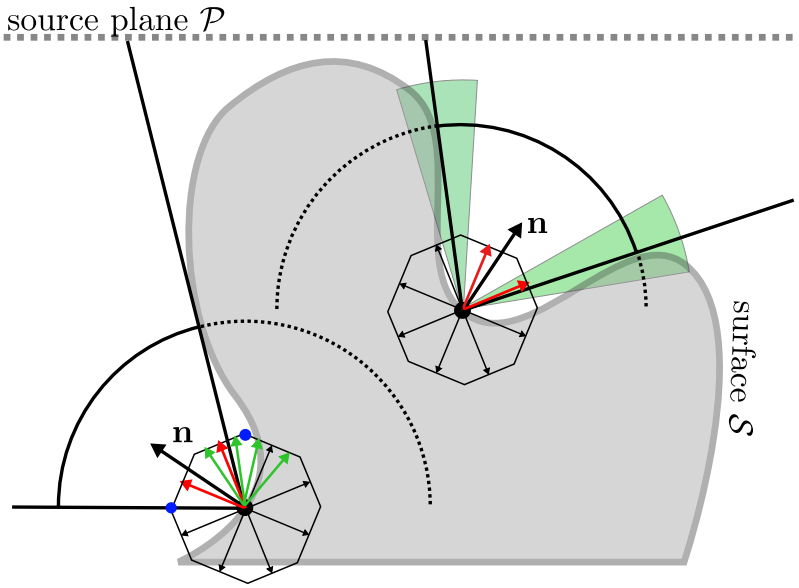


Figure 4.5: Two-dimensional illustration of the *bottom-up* visibility scheme for direct flux calculation for two surface points. The search directions are colored according to the source visibility (red = visible, black = obscured). In three dimensions, the search directions are towards the centers of the triangles of the icosahedron. The green solid angles indicate the neighborhood of the aperture boundary. The blue points are identified as aperture boundary points which are used to define the regions for re-evaluation on the next level. The green arrows indicate the refined search directions near the aperture boundary.

After the detection of the visible directions using the adaptive visibility scheme, the integration is performed with the resolution of *maxlevel* to provide the same integration accuracy compared to the non-adaptive sampling on *maxlevel*.

Algorithm 1 describes the adaptive visibility sampling for an integration point on the surface including the integration on the maximum refinement level.

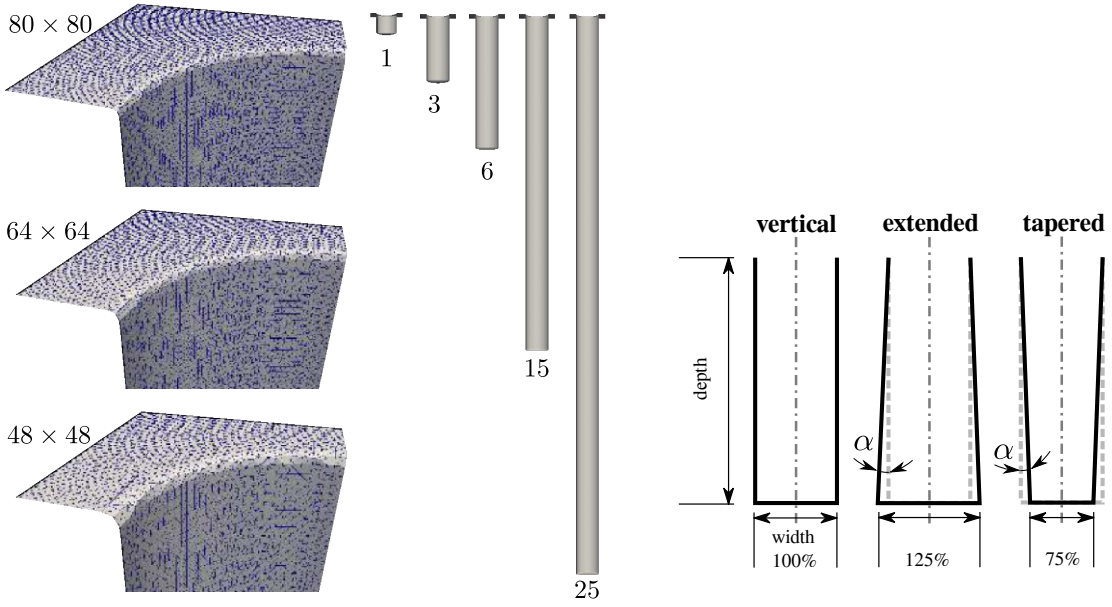
Algorithm 1: *Bottom-up* direct flux calculation at surface position i using the proposed adaptive visibility scheme and integration on the maximum refinement level.

```
Input  : minlevel, maxlevel
Output: flux at surface point i
for n = minlevel to maxlevel do
    if n == minlevel then
        set all directions to be active on level n;
        set all visibility information to false level on n;
    foreach direction d on level n do
        if d is active then
            trace ray into direction of d;
            if ray hit source then set visibility for d to true;
            if ray hit surface then set visibility for d to false;
    if n < maxlevel then
        foreach direction d on level n + 1 do
            set visibility for d according to parent on level n
        foreach vertex v on level n do
            if v is boundary vertex then
                foreach triangle t connected to v do
                    mark all children of t active on next level
    if n == maxlevel then
        foreach direction d on level n do
            if visibility of direction d is true then
                integrate flux contribution (cf. (4.5));
                add contribution to surface point i;
```

4.4 Evaluation Results

The algorithm is evaluated using a cylindrical hole with aspect ratios between 1 and 25 and resolutions (i.e., number of level-set grid cells) between 48×48 and 80×80 in the lateral directions (Figure 4.6a). This results in vertical resolutions of up to 1100, which represents computationally challenging evaluation cases. To investigate the sensitivity of the approach towards geometrical variations, the radius at the bottom is varied by $\pm 25\%$, leading to extended and tapered holes (Figure 4.6b).

The choice for the *minlevel* depends on the aspect ratio of the geometry and is illustrated in Figure 4.7: For a vertical geometry, the minimum angular resolution required to detect the opening aperture is shown and put in relation to the required number of subdivisions of the icosahedron. The *maxlevel* is set to 6 for all simulations, which corresponds to about $4 \cdot 10^4$ search directions per hemisphere. This choice allows at least one level of refinement for all tested aspect ratios between 1 and 25 (cf. Figure 4.7, dotted vertical lines).



(a) Resolutions and aspect ratios **(b)** Variations

Figure 4.6: (a) Triangulated surfaces of the vertical geometry for aspect ratios between 1 and 25. The opening is shown in detail (cut view) for three different horizontal resolutions. (b) Variations of the geometry.

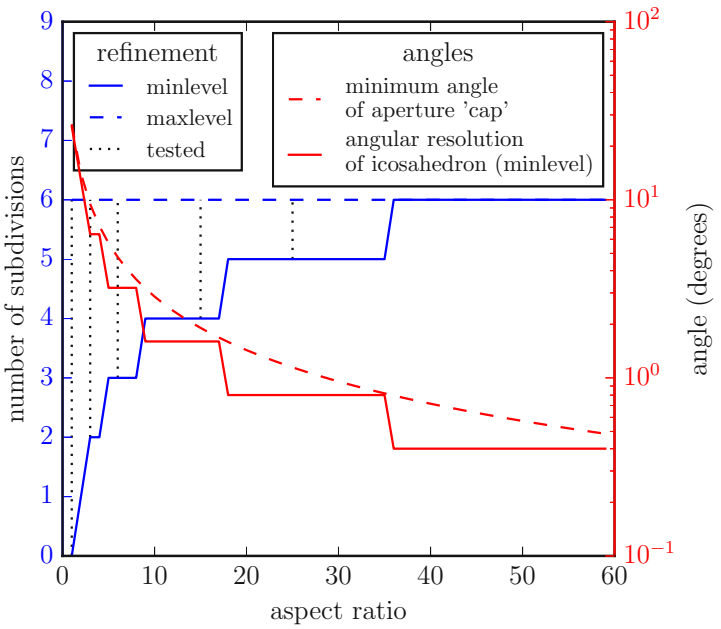


Figure 4.7: Plot of the minimum angular resolution (dashed red) and corresponding minimum subdivision level of the icosahedron (solid blue) to guarantee the detection of the aperture of the vertical geometry. The dotted vertical lines indicate the investigated aspect ratios 1, 3, 6, 15, and 25.

All results were produced using a highly, vertically focused[2] ($n = 100$) power cosine source distribution $\Gamma(\Theta) = \cos(\Theta)^n$ (cf. Figure 4.3). To verify that the accuracy of the integral indeed does not suffer, the flux rates are compared at various depths of the structure: The flux rates are identical when applying the proposed algorithm.

Figure 4.8 shows the cross section of a validation simulation with a linear surface velocity model $V_n = -F$ for the tapered structure (aspect ratio 6). In Figure 4.8a, the impact of a low maximum level of refinement is demonstrated: The position of the top surface is very inaccurate; this is due to the high derivatives of the source distribution (i.e., a very directed source) in conjunction with a low spatial sampling during the integration. The aperture is detected but the position of the bottom of the structure varies considerably due to an insufficient sampling of the aperture. In Figure 4.8b, using *maxlevel* = 5, the positions at the bottom are symmetric and close to the reference in Figure 4.8c. The position of the top surface still shows a noticeable difference to the analytical solution. In Figure 4.8c, also the top surface is very close to the analytical solution.

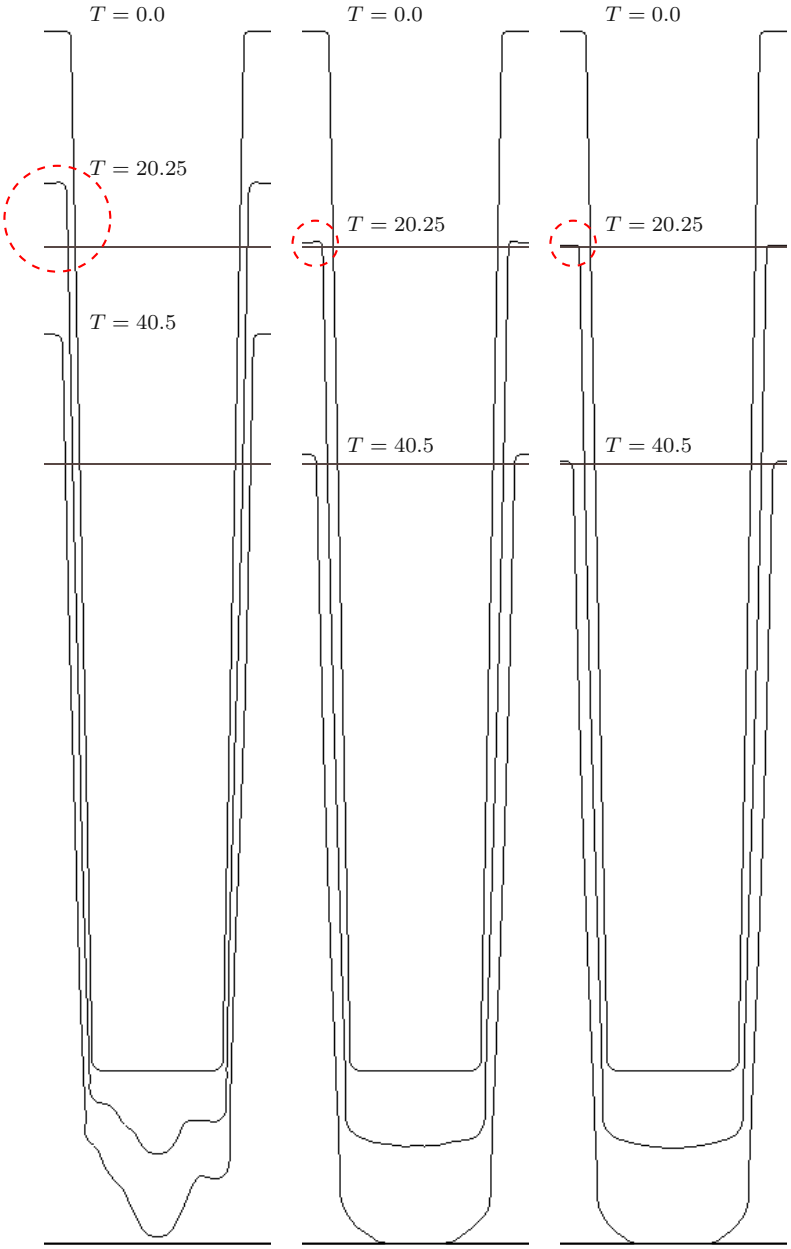
4.5 Performance Results

The average runtime for the parallelized (OpenMP) direct flux calculation (i.e., runtime consumed by `SurfaceratesIF.calcSurfRates()`, cf. Section 3.2) was tracked for various combinations of aspect ratios, spatial resolutions, and refinement levels. All performance measurements were performed on WS1. The obtained speedups when applying the presented adaptive sampling algorithm are summarized in Figure 4.9; the corresponding ratio of necessary visibility tests between the non-adaptive and the adaptive simulations are summarized in Figure 4.10.

If the visibility tests constituted the entire computational load and no overhead were introduced to maintain and access the visibility information on different subdivision levels of the icosahedron, the speedup would be expected to match the reduction ratio. Figure 4.9 and 4.10 reveal a factor between the speedup and the reduction: For one level of refinement, the factor is about 2 for all configurations; for 2 and 3 levels of refinement, the factor increases up to 4. The underlying reasons are the overhead of creating and accessing the visibility information (which increases with the number of refinement levels) and the computational load of the integration itself, which is performed on the *maxlevel* for the full aperture regions.

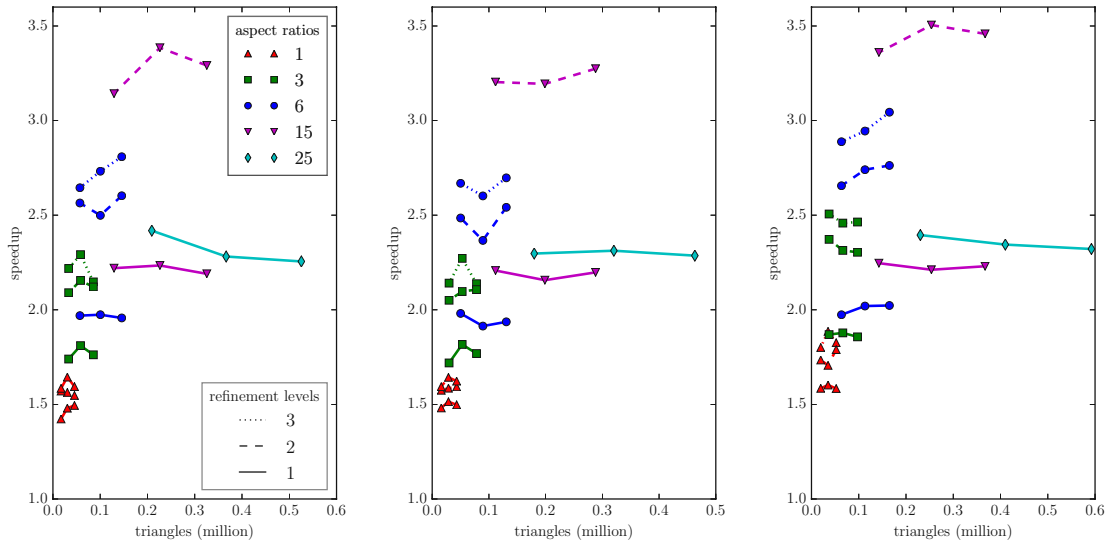
In summary, when only considering one level of refinement, a minimum speedup of 1.5 is obtained for aspect ratio 1. For surface meshes with more than $2 \cdot 10^5$ triangles (aspect ratios 15 and 25), the speedup is larger than 2 for all configurations.

[2]Highly vertically focused power cosine sources are used in *Process TCAD* to model vertically accelerated particles (ions).



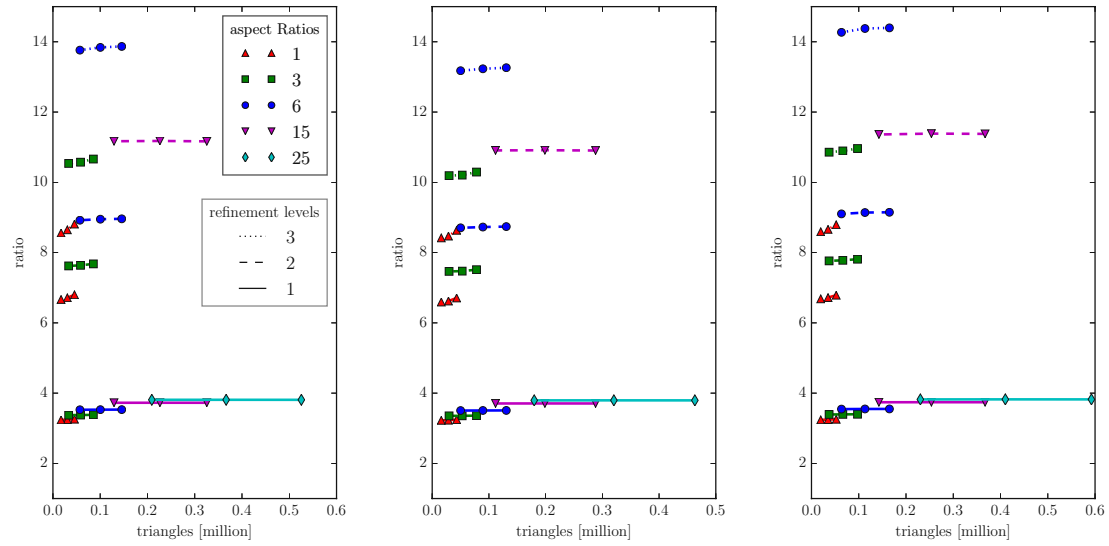
(a) *min*=3, *max*=3 **(b)** *min*=4, *max*=5 **(c)** *min*=5, *max*=6

Figure 4.8: Cross section of three-dimensional simulation results for a linear surface model $V_n = -F$ and for the tapered structure of aspect ratio 6 from $T = 0$ to $T = 40.5$ for different refinement settings (a,b,c). The straight, horizontal lines in the upper part (highlighted by red circles) indicate the analytical solution for a horizontal surface.



(a) Vertical **(b)** Tapered **(c)** Extended

Figure 4.9: Speedup obtained by applying the presented adaptive sampling scheme for different surface mesh sizes (x axes) and aspect ratios (symbols) and geometry variations (a-c). The line style indicates the number of refinements; *maxlevel* is 6 for all results.



(a) Vertical **(b)** Tapered **(c)** Extended

Figure 4.10: Ratios of necessary visibility tests for the full sampling on *maxlevel* and the adaptive sampling scheme (configurations identical to Figure 4.9).

4.6 Summary

It is shown that the *bottom-up* adaptive visibility sampling scheme is able to sustain the accuracy of the direct flux integrals while reducing the integration time by 50% for larger meshes ($> 2 \cdot 10^5$ triangles), as long as all opening apertures are captured by the minimum level of refinement.

When using more than one level of refinement, the ratio between integration time and ray tracing time limits the obtainable speedup, as the integration is always performed on the maximum level of refinement.

Chapter 5

Ray-Surface Intersection Tests for Particle Transport

The particle transport calculation in either *bottom-up* or *top-down* schemes relies on a large number of ray-surface intersection tests (cf. Section 2.2.2). The result of an intersection test is the distance from the origin of the ray to the closest intersection with the surface in the direction of the ray. Figure 5.1 illustrates two choices for the representation of a surface: An implicit representation (e.g., a signed-distance field) and an explicit representation (triangle mesh) of a sphere.

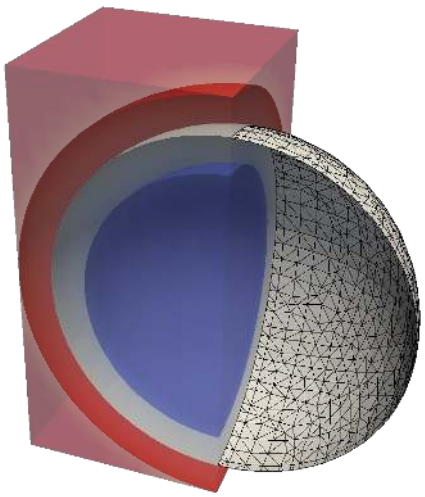


Figure 5.1: Illustration of two surface representations of a sphere: Left: Implicit representation using a signed-distance field. The blue and red surfaces are the isosurfaces delimiting the narrow-band of 3 grid cells around the interface to the inside (blue) and outside (red). The gray surface is the zero-level-set. Right: Triangle mesh extracted from the zero-level-set.

If an explicit representation of the surface is used, the intersection test identifies the closest intersection point and the corresponding primitive, e.g., a specific triangle out of a triangle mesh. A straightforward implementation is to test each primitive of a mesh for an intersection with the ray and store the resulting distance to the intersection point. Finally, the closest intersection point and the corresponding

primitive is reported. The primitives need not necessarily form a closed polygonal surface, e.g., in [1] tangential disks are used and in [19] spheres are used to represent the surface.

For an implicit representation, e.g., a signed-distance field, the intersection point is a coordinate (on the isosurface) between the vertices of the grid holding the signed-distance field. A straightforward implementation is to advance a test point in steps along the ray direction. The step length is thereby equal to the absolute value of the signed-distance field at the location of the test points. The signed-distance value is then re-evaluated at the new position of the test point. The procedure is repeated until the signed-distance value changes its sign or the absolute value falls below a threshold value. The last position of the test point is then reported as the intersection point. This procedure is referred to as *sphere tracing* [82].

In applications requiring a high throughput of ray-surface intersections, e.g., rendering or non-imaging optics, typically a spatial data structure is constructed from the surface representation to accelerate the intersection tests. The most common data structure is a bounding volume hierarchy (BVH), which reduces the number of necessary intersection tests by grouping primitives hierarchically into bounding boxes.The intersection tests are then performed with the bounding boxes. Only if a bounding box is intersected by a ray, the hierarchically lower bounding boxes (or contained primitives) are tested for intersection. Hardware-tailored ray tracing libraries (cf. Section 2.2.2.1) originally designed for computer graphics applications provide highly-optimized ray tracing kernels for common modern computing platforms. As target applications commonly include real-time rendering of dynamic geometries, the libraries also focus on optimizing the performance of the BVH construction. Most libraries solely support single-precision arithmetics; this allows to fully utilize SIMD instructions on modern CPUs and to utilize the high single-precision processing power of GPUs.

In a feature-scale process simulator, the surface is typically represented as a narrow-band level-set (cf. Section 2.1). The most widely used approach, and thus the frame of reference for the here presented evaluations, is to perform the ray casting on the implicit representation, i.e., the level-set function. An advantage is that no explicit representation of the surface has to be extracted in each time step. However, the consequential lack of a closed polygonal explicit surface representation is disadvantageous, as the surface topology in the neighborhood of a surface point is not readily accessible, which is demanded by spatially adaptive approaches for the calculation of the particle transport (cf. Section 6.1).

In the following, an in the course of this work developed approach is presented which extracts an explicit polygonal mesh in each time step of a feature-scale simulation [80]. This mesh is then used for the ray casting during the particle transport. The underlying motivation is to utilize the higher throughput of ray-surface intersections during the particle transport; however, the overhead introduced by mesh extraction from an implicit surface representation must be considered. The ray casting is performed using single-precision arithmetic.

First, the accuracy of ray casting using single- and double-precision floating point representations is compared using a generic test scenario. Then, the performance difference for single-precision ray casting is compared using two highly optimized

open-source libraries for ray casting on implicit and explicit surfaces. Finally, the newly developed approach is presented in more detail and an evaluation of the performance is presented for a simple deposition test case.

5.1 Single-Precision Ray Casting

The sufficiency of single-precision arithmetics depends on the numerical method and the further use of the result. The methods to calculate the particle transport (cf. Section 2.2) use ray casting to detect ray-surface intersections along a certain direction. The resulting information is binary and therefore the influence of the arithmetic precision of the ray casting is isolated.

To evaluate the applicability of single-precision ray casting for the ray-surface intersections during the particle transport, the angular resolutions which are achieved for single- and double-precision ray casting are compared. A spherical mesh (double-precision vertex coordinates), generated by subdividing an icosahedron 6 times (cf. Section 4.1, i.e., about $4 \cdot 10^4$ vertices), serves as a reference mesh. The radius of the reference mesh is scaled from $r = 10^{-16}$ to $r = 10^{18}$ and rays are traced from the origin towards each vertex of the mesh. Figure 5.2 conceptually illustrates the spherical reference mesh at different scales and the corresponding rays towards the vertices of the mesh. The distance between the intersection point (found by ray casting) and the vertex coordinate on the reference mesh is computed for all vertices. The ratio of this distance and the radius r provides information about the maximum angular resolution of the intersection test.

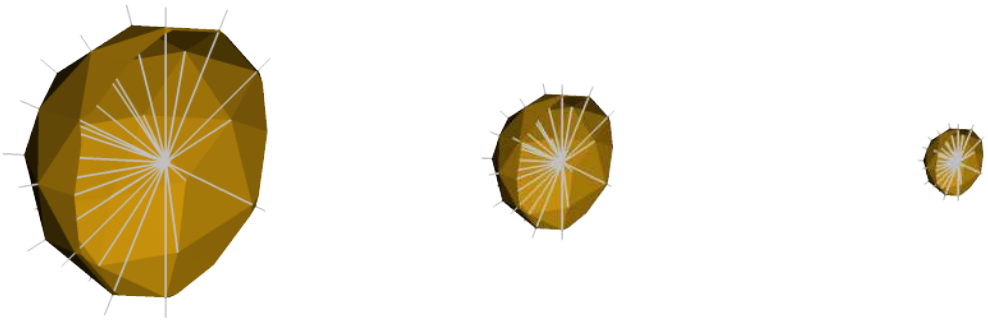


Figure 5.2: Conceptual visualization of the geometric configuration of the ray casting precision evaluation: The cross section of the spherical mesh is shown (for different scales) together with the rays which start at the origin and point into the directions of each vertex of the spherical mesh. The distance between the intersection position of each ray with the mesh (found by ray casting) and the corresponding vertex coordinate are calculated for single- and double-precision ray casting.

The *NanoRT* library [25] is used for single- and double-precision ray casting; the *Embree* library [24] is additionally used for single-precision. In Figure 5.3, the results for the maximum distance d_{max} and the distance normalized to the radius $d_{norm} = d_{max}/r$ are plotted over the radius r of the reference sphere. The normalized distance is constant for single- and double-precision, respectively; the constant values are $\approx 10^{-7}$ and $\approx 5 \cdot 10^{-16}$ falling in the range of significant digits for single-

and double-precision, respectively. For $r < 10^{-12}$ and $r > 10^{13}$ no intersection is found for single-precision ray casting using *NanoRT*; the same holds for *Embree* where the lower limit is $r < 10^{-7}$. The angular resolution of subdivided icosahedrons (cf. Section 4.1) are indicated with dotted lines. The results reveals that

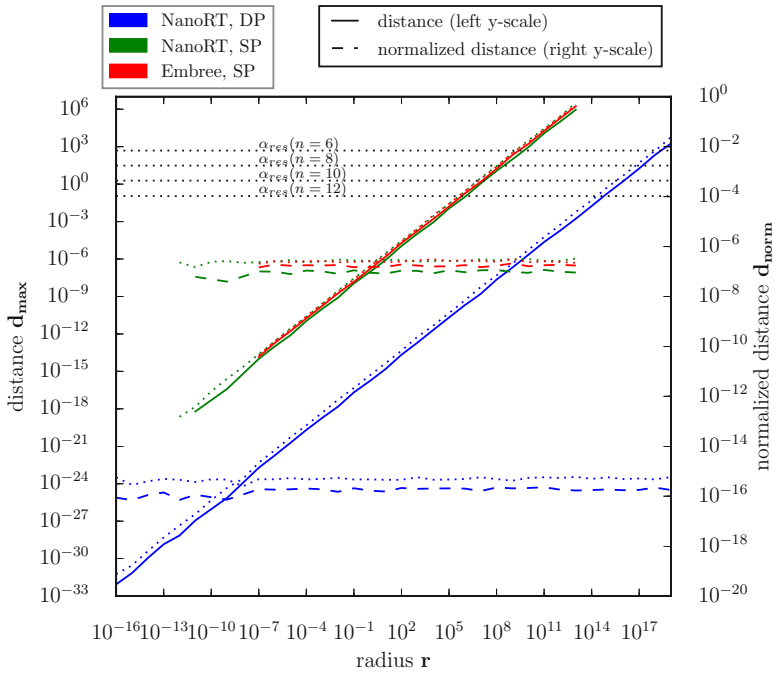


Figure 5.3: Evaluation of the angular resolution using single-precision (green, red) and double-precision (blue) ray casting. The maximum distance d_{max} between the intersection point (found by ray casting) and the reference point is shown as solid line and the scale on the left. The normalized distance $d_{norm} = d_{max}/r$ is shown as dashed line and the scale on the right. Two lines are shown for each result which correspond to two different ways of calculating the intersection point with the mesh: (a) using a scaling of the direction of the ray with the distance of the intersection (solid for d_{max}, dashed for d_{norm}) or (b) using the coordinates of the intersected primitive combined with the local intersection coordinates (dotted). Additionally, the angular resolutions of a subdivided icosahedron are shown as horizontal dotted lines for up to $n = 12$ subdivisions.

the achieved angular resolution is more than two decades smaller than the angular resolution of a 12 times subdivided icosahedron for both, single-precision and double-precision ray casting.

In conclusion, the arithmetic precision achieved with single-precision ray casting is sufficient to calculate the ray-surface intersection tests in practical three-dimensional feature-scale process simulations: For *bottom-up* flux calculation schemes, the angular resolution is sufficient even for the highest practical spatial sampling resolutions; the same holds for *top-down* flux calculation schemes. For instance, considering a cylindrical hole of aspect ratio 1000, i.e., depth 1000 and diameter 1: Taking the area of the opening aperture $A_o = 0.5^2\pi$ and approximating a ray direction starting from the bottom of the structure with coverage

$A_{ray} = (1000 \cdot 10^{-7}/2)^2\pi$ on A_o, the aperture can be sampled with $A_o/A_{ray} = 10^7$ ray directions. The bottom of the hole can consequently be sampled with 10^7 rays starting at a single position on the source plane.

5.2 Ray Casting Performance for Non-Imaging Applications

In rendering applications, the ray tracing performance is commonly reported in achieved frames per second for a given scene and perspective. An alternative metric, and suitable for topography simulations in *Process TCAD*, are the traced rays per second, usually denoted in million rays per second (Mrays/s). The actual computations assigned to the tracing of a single ray are defined by the rendering algorithm and may include followup computations, e.g., to calculate shading. The following benchmark solely aims at evaluating the performance of the *raw* intersection test (i.e., the calculation of the closest intersection point with a surface) for spatially incoherent rays.

The surface is a unit sphere with $r_{unit} = 1$ centered at the origin for all configurations in the following. The origins of the rays are distributed on another sphere with $r_{org} = r_{unit} + d$; this leads to rays which start inside the surface for $d < 0$ and rays which start outside the surface for $d > 0$. At each origin, rays are traced towards the centroids of the triangles of a subdivided icosahedron (cf. Section 4.1). Figure 5.4 illustrates this benchmark scheme conceptually for two-dimensions.

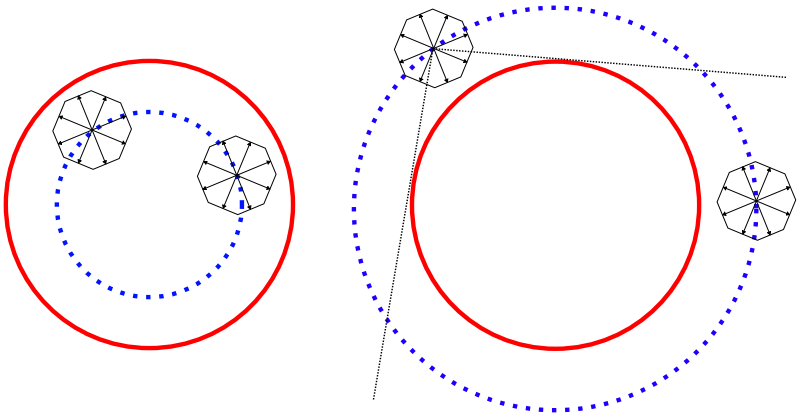


Figure 5.4: Conceptual two-dimensional illustration of the benchmark scheme: The red circle represents the unit sphere and the blue dotted circles represent the spheres on which the ray origins are distributed. The black arrows indicate the scheme for the ray directions which are defined using a subdivided icosahedron. The left and right side illustrate the configurations $r_{org} < r_{unit}$ and $r_{org} > r_{unit}$, respectively.

The performance of two open-source libraries from the field of computer graphics is compared: *OpenVDB* [31] is used for implicit, and *Embree* [24] for explicit surfaces. The rays are traced against a narrow-band level-set representation of a sphere with radius $r_{mesh} = 1$ using *OpenVDB*'s `LevelSetRayTracer` and against a triangulated mesh (extracted from the level-set) using *Embree*. The implicit mesh is

represented with *OpenVDB*'s default tree configuration `Tree4<float,5,4,3>` and a narrow-band half-width of 3 level-set grid spacings d_{vox}. The `for` loop which iterates over the ray origins is OpenMP-parallelized. All performance measurements were performed on WS1. The benchmarks were performed using *Embree* 2.12, *OpenVDB* 4.0, and compiled using gcc 6.1.1. Table 5.1 summarizes the parameter range of the performance analysis.

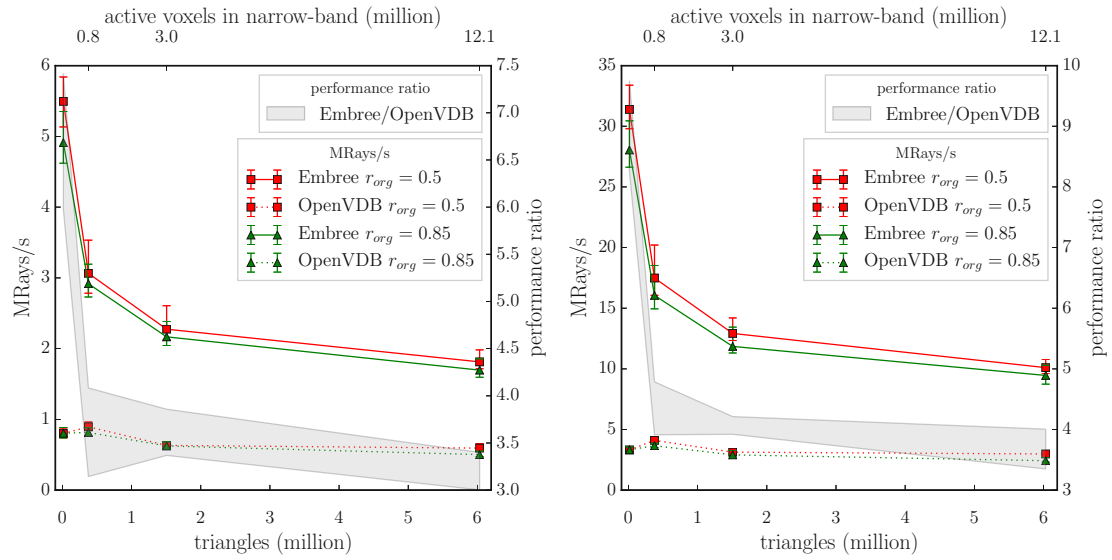
Parameter	Values
Number of threads	1, 2, 4, 5, 6, 8
Subdivisions for search directions n_{trace}	1, 2, 3, 4, 5
Radius of origins r_{org}	1.5, 1.15, 0.85, 0.5
Voxel size d_{vox}	0.05, 0.01, 0.005, 0.0025
Dependent Parameter	
Number of active voxels $f(d_{vox})$	30K, 0.8M, 3.0M, 12.1M
Number of triangles $f(d_{vox})$	15K, 0.4M, 1.5M, 6.0M
Number of search directions $f(n_{trace})$	80, 320, 1280, 5120, 20480

Table 5.1: Parameter variations used in the performance comparison (K = thousand, M = million). An active voxel is a level-set grid cell in the narrow-band level-set around the surface.

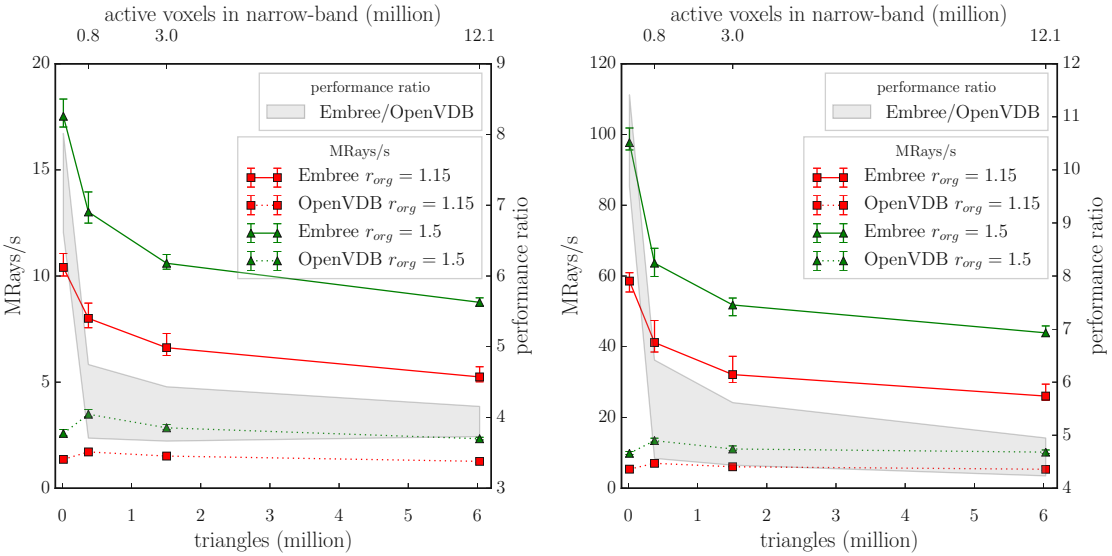
Fig 5.5 compares implicit and explicit ray casting performance for different ray origins r_{org} and different surface resolutions. The limits of the achieved performance with 8 threads for *Embree* are about 100 Mrays/s ($r_{org} = 1.5$, 15K triangles, Figure 5.5c) and about 10 Mrays/s ($r_{org} = 0.85$, 6.0M triangles, Figure 5.5b). For the implicit ray casting using *OpenVDB*, the limits are about 13 Mrays/s ($r_{org} = 1.5$, 0.4M triangles, Figure 5.5c) and 2 Mrays/s ($r_{org} = 0.85$, 6.0M triangles, Figure 5.5b).

The resulting performance gain is between 3 and 6 for all possible combinations of the parameters in Table 5.1, excluding low spatial resolutions (i.e., less than 0.4 million triangles). The performance gain is higher for the multi-threaded runs, the main reason being the higher speedup for *Embree* in the hyper-threading regime (cf. Figure 5.6). For low spatial resolutions (less than 0.4 million triangles) the performance ratio is increasing. For high spatial resolutions (i.e, more than 1 million triangles) the multi-threaded performance ratio is ≈ 3.5 for $r_{org} < 1$ and ≈ 4.5 for $r_{org} > 1$, i.e., *Embree* profits more than *OpenVDB* if a large portions of the rays do not intersect the geometry at all.

Fig 5.6 plots the achieved speedup for the parallelized `for` loop (which iterates over the ray origins) for explicit and implicit ray casting. Both parallelizations show nearly an ideal speedup for 2 threads. The speedup spreads for 4 threads with a minimum speedup of 2.7 (*Embree*) and 2.0 (*OpenVDB*). The speedup for 8 threads is up to 6 for *Embree* and up to 5 for *OpenVDB*. The parameter combinations which show nearly no speedup between 2 to 4 threads (cf. Figure 5.6b) were identified to have a low hit ratio (number of hits/number of traced rays < 0.13) and a large voxel size ($d_{vox} >= 0.01$). For 5 and more threads (entering the hyper-threading regime) this influence is compensated leading to an overall speedup between 3 and 4 for 6 threads.

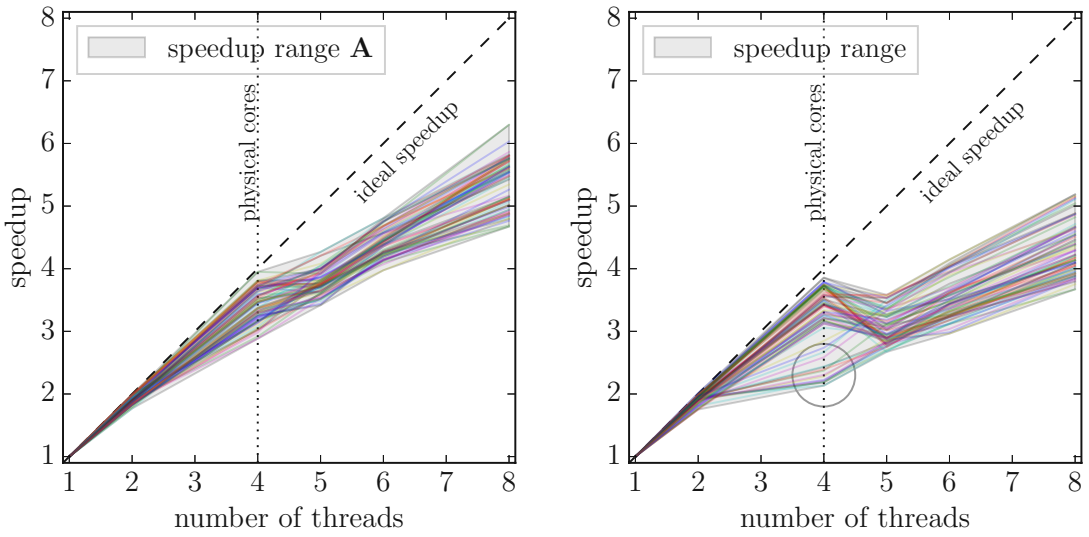


(a) 1 thread, $r_{org} < 1$ **(b)** 8 threads, $r_{org} < 1$



(c) 1 thread, $r_{org} > 1$ **(d)** 8 threads, $r_{org} > 1$

Figure 5.5: Performance comparison 1 and 8 threads between ray casting on the implicit surface (using *OpenVDB*) and on the explicit surface (using *Embree*). (a)-(b): Ray origins $r_{org} < 1$. (c)-(d): Ray origins $r_{org} > 1$. The top axis plots the number of active voxels in the narrow-band representation of the surface while the bottom axis labels the corresponding number of triangles in the extracted mesh. The error bars show the spread in performance when varying the number of search directions for each origin according to Table 5.1. The filled gray area is the range of the performance ratio of the explicit approach (*Embree*) over the implicit approach (*OpenVDB*).



(a) *Embree* **(b)** *OpenVDB*

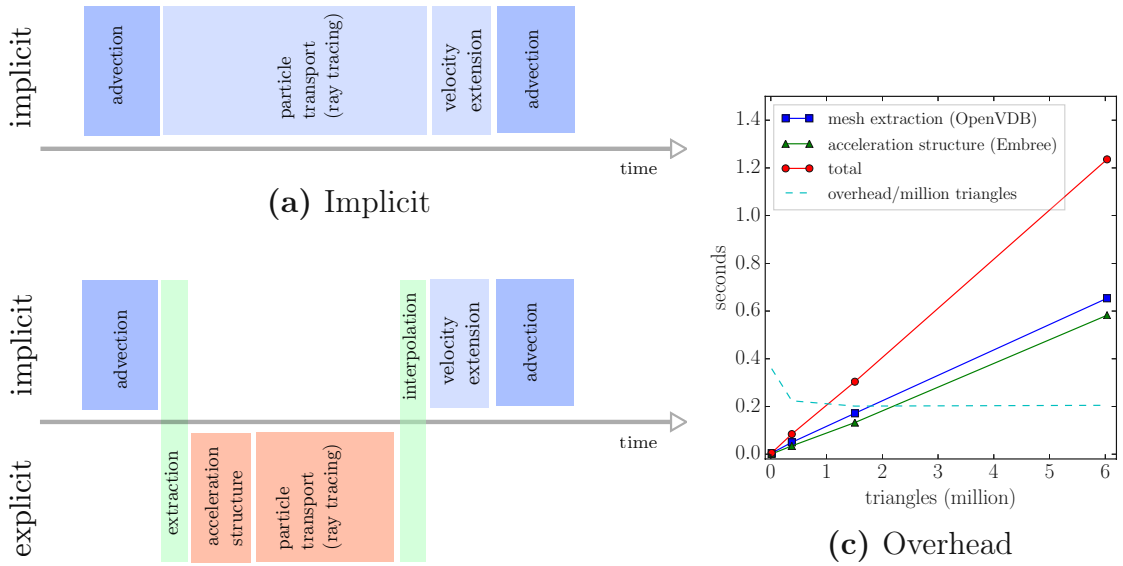
Figure 5.6: Speedup for the OpenMP-parallelized ray casting with *Embree* (a) and *OpenVDB* (b) on WS1. The speedup range represents limits resulting of the full parameter range (cf. Table 5.1). The dashed line indicates an ideal speedup and the dotted line marks the number of physical cores in the system. The circle in (b) marks a group of combinations with nearly no speedup between 2 and 4 threads.

5.3 Temporary Explicit Meshes for Flux Calculation

Based on the results from the previous sections a scheme is introduced which aims to accelerate the particle transport calculation of a level-set-based process simulation. It bases on the extraction of a temporary explicit surface mesh (from the level-set) in each time step of the simulation. Figure 5.7a and 5.7b illustrates the sequence of computational tasks in a time step of a level-set-based process simulation. The scheme introduces new computational tasks for each time step, namely

- the extraction of the explicit surface mesh from the level-set,
- the generation of an acceleration structure (for efficient intersection tests) from the extracted mesh, and
- the interpolation of the surface rates from the explicit mesh to the positions of the level-set grid (computationally negligible).

Using the libraries benchmarked in Section 5.2 the overhead introduced by the approach is estimated. Figure 5.7c plots the runtime on WS1 (8 threads) for the generation of the temporary explicit mesh and the construction of the acceleration



(a) Implicit

(b) Implicit/explicit

(c) Overhead

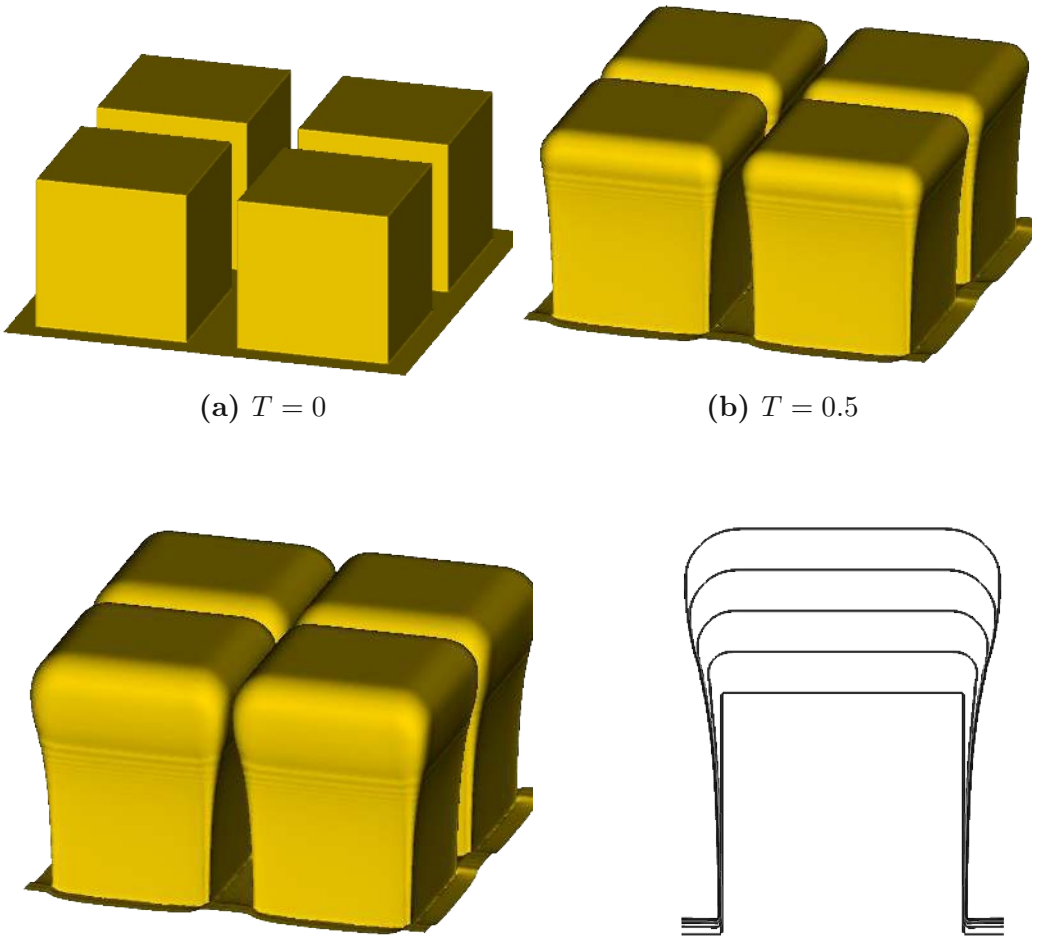
Figure 5.7: Difference in the sequence of computational tasks in a time step between the fully implicit scheme (a) using a level-set to represent the surface and the implicit/explicit scheme (b) using a temporary explicit mesh to represent the surface during the particle transport calculation. In (c), the overhead introduced by the extraction of the explicit mesh (using *OpenVDB*) and the initialization of the acceleration structure (using *Embree*) is shown for different resolutions of the unit sphere.

structure, when using *OpenVDB* to represent the level-set and to extract the mesh, and the acceleration structure of *Embree*. Different resolutions of the unit sphere (analog to the benchmarks above) are tested; the maximum runtime is identified with a total of less than 1.4 seconds for a mesh with about 6 million triangles. The overhead per million triangles is about 0.2 seconds for meshes with more than 0.4 million triangles.

Performance Evaluation

The simulation platform described in Section 3 is used to implement a simple deposition test case: A cube with edge length 1.5 is centrally placed in a 2×2 domain with periodic boundary conditions. The direct flux F from an ideal-diffuse source is calculated and a simple linear deposition $V_n = F$ is used as surface velocity model. The direct flux is calculated using a *bottom-up* scheme with a 4 times subdivided icosahedron (cf. Section 4.1) to sample the spherical directions. The lateral level-set resolution is set to 128×128 and the resulting explicit surface meshes for the initial and final geometry count about 150K and 250K triangles, respectively. Figure 5.8 illustrates the initial, intermediate, and final geometry of the deposition simulation.

Figure 5.9 shows the runtime of the main computational tasks throughout the simulation. The runtime for the calculation of the surface rates (i.e., the direct flux calculation in this case) is dominating for both simulations. Nevertheless, the

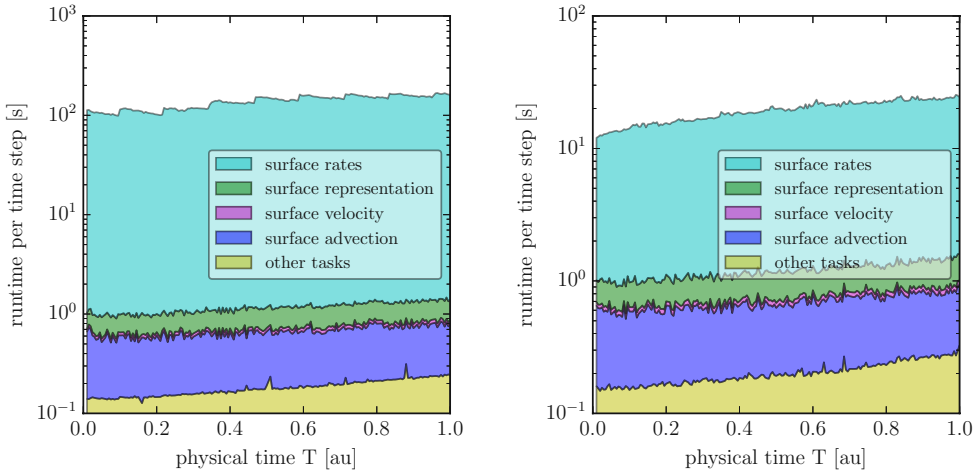


(a) $T = 0$ **(b)** $T = 0.5$

(c) $T = 1.0$ **(d)** Cross sections

Figure 5.8: Results of the the deposition test case for lateral level-set resolution 128×128. (a)-(c): Quadrupled visualization of the initial, intermediate, and final three-dimensional surface. (d): Cross sections for time $T = 0.0, 0.25, 0.5, 0.75, 1.0$.

speedup is about 7 to 9 when using explicit ray tracing. The speedup is in accordance with the estimates of the generic benchmark in Section 5.2. However, differently to the benchmark above, the ray origins are located near the surface, i.e., they start inside the narrow-band of the level-set. Considering the geometry visualized in Figure 5.8c, it is apparent that most rays start in the narrow-band and intersect the narrow-band. This configuration is not considered in the benchmark above, where all ray origins are located at some distance to the narrow-band.



(a) Implicit ray tracing **(b)** Explicit ray tracing

Figure 5.9: Runtime of the main computational tasks throughout the deposition test case when using (a) implicit ray tracing, and (b) explicit ray tracing during the direct flux calculation.

5.4 Summary

Single-precision ray tracing is attested sufficient accuracy, for *top-down* and *bottom-up* approaches for the particle transport in practical simulation scenarios.

The ray casting performance when using *OpenVDB* for implicit surfaces and *Embree* for explicit surface representations is studied using a generic test for non-imaging application. The performance gain is at minimum a factor of 3 for a wide range of scenarios.

An approach to perform the ray-surface intersections not on the level-set-based implicit representation of the surface but on a temporary explicit mesh was presented and compared. The performance gain in a deposition test case using a *bottom-up* approach to calculate the direct flux is a factor between 7 and 9 using the benchmarked libraries.

Chapter 6

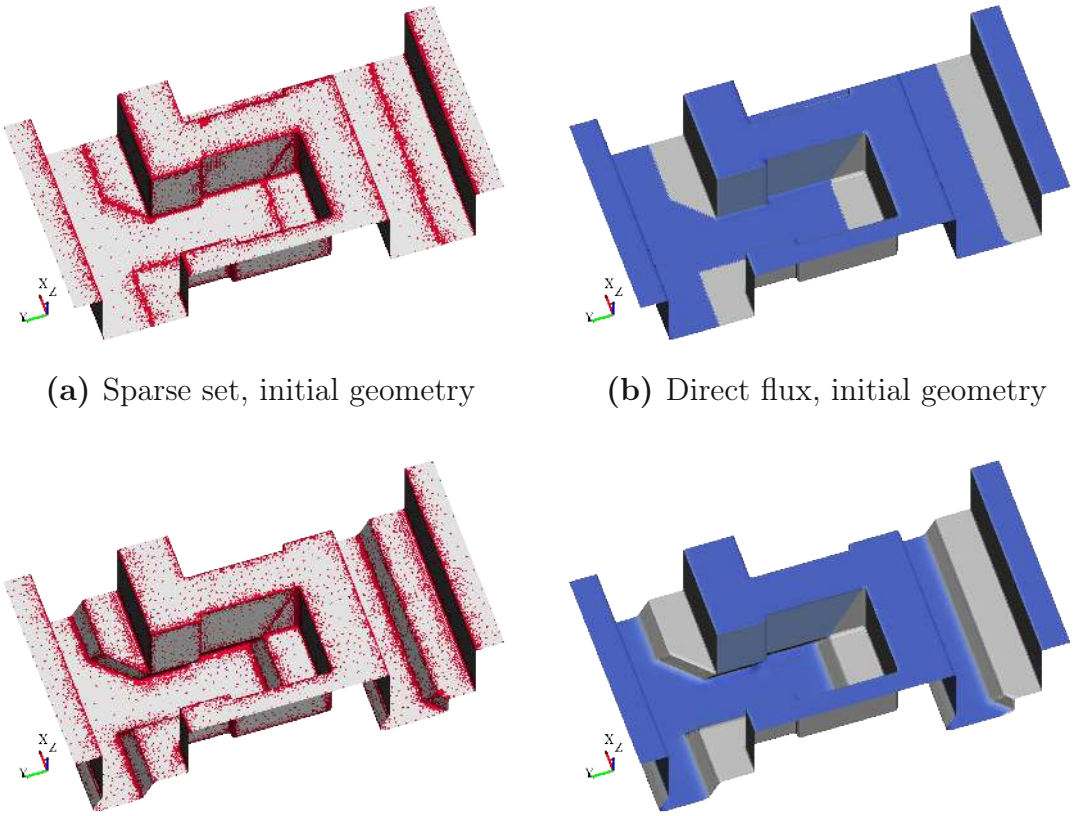
Sparse Evaluation of Surface Velocities

A constant spatial resolution throughout the domain is a straightforward choice for feature-scale process simulation. An advantage is that numerical schemes for advection and extraction algorithms can be applied without the need to consider multiple resolutions. Another advantage is that geometric features are represented in the full domain with maximum accuracy. For high spatial resolutions, this results in a highly resolved surface representation which is used during the particle transport calculation. In a straightforward approach, the surface rates are evaluated using the full resolution of the surface. This represents a valid and robust approach as the surface advection is conducted with maximum spatial accuracy. The particle transport is typically dominating the runtime of the simulation in such cases.

The material surfaces potentially experience very different advection rates, depending on the simulated process and the geometric configuration. For instance, a large portion of the fully exposed horizontal surface of a photoresist is advected with a constant vertical velocity. Other regions of the surface potentially exhibit locally very different advection rates, e.g., in regions near the transition from a vertical to horizontal surface orientation.

Considering the resulting different requirements for spatial resolutions, one approach is to reduce the overall computational complexity by locally reducing the spatial resolution (of the level-set) in regions where it is admissible without sacrificing the advection accuracy. Within the scope of the example discussed above, a lower spatial resolution is used for the top surface of the photoresist. Approaches based on a local spatial reduction of the resolution cannot solely rely on the geometrical properties of the surface: Local variations of the advection rate might also occur on planar material surfaces due to shadowing from remote regions of the surface. That is, depending on the local surface advection rates, the level-set resolution has to be re-adopted accordingly, e.g., to capture features of the geometry which develop in previously planar regions.

The approach presented in the following is based on a constant spatial resolution of the level-set function [83]. The computational complexity is reduced by selecting a sparse set of points on the surface. The surface rates are evaluated only on this sparse set leading to a reduction of the computational demand of the direct flux calculation.



(a) Sparse set, initial geometry **(b)** Direct flux, initial geometry

(c) Sparse set, etched geometry **(d)** Direct flux, etched geometry

Figure 6.1: *Top layer* of a two-material etching simulation for a focused source with direction $[0.5, 0.5, -1]$ to demonstrate the result of the iterative partitioning scheme. The resulting sparse set of points and corresponding direct flux for the initial geometry (a) (b) and geometry and at a later etched stage (c) (d) is visualized. The blue color in (b) and (d) encodes the received direct flux, whereas gray corresponds to shadowed regions, i.e., zero flux.

This sparse set of points is generated according to application-specific requirements with an iterative partitioning scheme: The local geometrical properties and local deviation of the direct flux are used to locally define the resolution of the sparse points. The flux rates at the sparse points are also applied in the neighborhood which is assigned to each sparse point. This constant extrapolation is then "diffused" to obtain an approximation of a linear interpolation between these sparse points. Figure 6.1 demonstrates the approach for a structure etched by the direct flux from a tilted focused particle source with direction $[0.5, 0.5, -1]$. The red triangles on the surface of Figure 6.1a and 6.1c depict the sparse set of points for which the surface rates are evaluated. Figure 6.1b and 6.1d illustrate the corresponding direct flux.

In the following, first the iterative partitioning scheme and the subsequent interpolation are introduced in detail. Then, the method and its performance is evaluated using different resolutions of a simple etching test case.

6.1 Iterative Partitioning Scheme

The scheme presented in the following provides a robust method to reduce the number of necessary evaluation points for which the surface rates are calculated. From a dense set of evaluation points given by the resolution of an explicit surface mesh, a subset of points is selected using an iterative partitioning scheme. The scheme is controlled by a freely definable refinement condition, allowing to adopt the method for different application-specific requirements (cf. Section 6.3).

The *dense* set of evaluation points is defined as the set of all triangle centroids. Algorithm 2 is used to iteratively select a *sparse* subset of evaluation points depending on (a) the maximally globally allowed edge distance (d_{max_0}) between two points in the subset, (b) an array of maximally allowed edge distances for each point in the dense set where each entry is between 0 and d_{max_0}, and (c) a refinement condition. The refinement condition defines in each iteration and for each point in the sparse set, if additional points in the surrounding should be added to the sparse set. Algorithm 2 assigns one of the sparse points to each of the points in the dense set: This way, a neighborhood is formed from all points with the same sparse "parent". This neighborhood is referred to as *patch* in the following. The patches are the "spacers" between the points in the sparse set and are used to efficiently identify neighbors in the sparse set and to generate the initial guess for the Jacobi solver discussed in Section 6.2. As will be discussed later in more detail, the refinement condition used in Section 6.3 is based on a fixed threshold for the angular deviation of the surface normal and the deviation of the direct flux (from a source) between a sparse point and its sparse neighbors. Details for the functions in Algorithm 2 can be found in Appendix C.2.

Figure 6.2 illustrates the individual stages of the algorithm using a small, regular triangle mesh. After the refinement is completed for all patches where the refinement condition evaluates to true, the refinement condition is re-evaluated for all sparse points. Subsequently, the refinement is repeated with $d_{max_2} = d_{max_1}/2$, continuously leading to a bisection of the maximal edge distance between sparse points on the patch. The algorithm is terminated either because the refinement condition evaluates to false for all sparse points or $d_{max_i} = 1$, corresponding to a patch consisting of only one triangle. If the refinement condition depends on the surface velocity at the sparse points, the surface model must be evaluated for the newly added sparse points in each iteration. After completion, Algorithm 2 provides a sparse set of points with corresponding sparse neighbors and patch information.

How well the edge distances between the sparse points map to arc length distances on the triangular mesh depends on the uniformity of the mesh with respect to triangle shape and size. Only with a mesh consisting of triangles with comparable size and quality the algorithm will produce "convex" patches (convex with respect to the projected polygon constructed of the outermost centroids).

Figure C.3 in Appendix C.1 demonstrates the resulting set of sparse points for each iteration when using $d_{max_0} = 16$ for the two-material geometry introduced in Figure 6.1. The refinement condition (6.2) is employed and the sparse set of points is initialized with the points near the material interfaces.

Algorithm 2: Adaptive decimation of evaluation locations on a triangular surface mesh.

Input: d_{max0}, distTarget[i], RefinementCondition(i)
Output: active[], sparseNeighbors[], patches[]
Algorithm

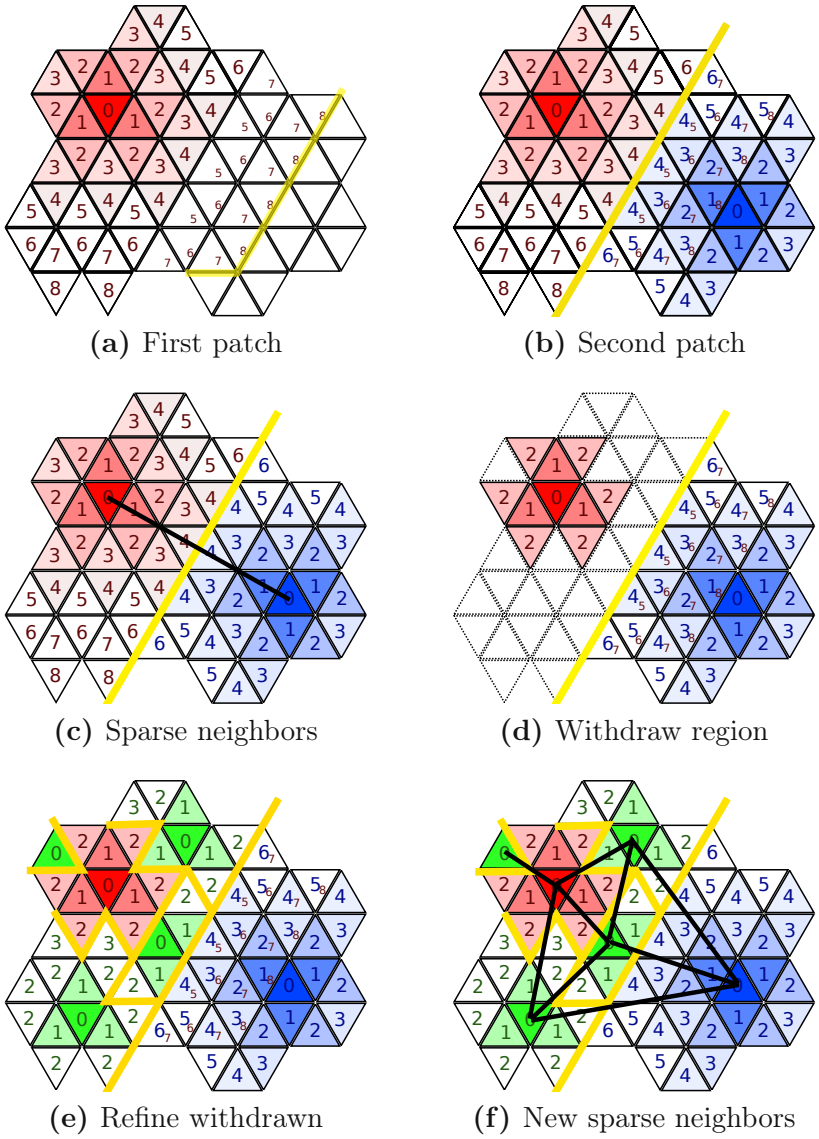
```
withdrawn[N_tri] = true; reflagged[N_tri] = false; active[N_tri] = false;
distance[N_tri] = d_max0; parent[N_tri] = -1;
patches[] = empty map(activeIndex,patchIndices);
sparseNeighbors[] = empty map(activeIndex,activeNeighbors);
indices[N_tri] = iota(0,N_tri);
FlagTriangles(indices, d_max)
RebuildPatches();
EvaluateSurfaceModel(for all active indices)
for n=1...log2(d_max0) do
    reflagged[N_tri] = false;
    withdrawn[N_tri] = false;
    numNewPatches = 0;
    foreach patch in patches do
        i_active = patch.activeIndex;
        if RefinementCondition(i_active) == true AND reflagged[i_active]
          == false then
            numNewPatches += RefinePatch(i_active, d_max0/2^n);
    if numNewPatches == 0 then
        break;
    RebuildPatches();
    EvaluateSurfaceModel(for all newly active indices)
```

6.2 Interpolation Between Sparse Points

Inherent to its construction method, the sparse set of points and the connections between sparse neighbors do not necessarily allow to construct a sparse mesh covering the complete original surface, which could be used for interpolation. To provide a robust, non-supervised, and computationally efficient interpolation between the sparse points, a constant extrapolation inside the patches using the corresponding values at the origins is performed, i.e., the values of the sparse points are assigned to the full corresponding patch. The properties of Laplace's equation (6.1) and the error diffusion properties of the Jacobi method [84] are used to smooth the "jumps" in the constant extrapolation and to approximate a linear interpolation between the sparse points:

(a) In one dimension, the solution of Laplace's equation (6.1) is equivalent to a linear interpolation between a sparse set of points when using the sparse set as Dirichlet boundary conditions and modeling the boundaries of the domain as Neumann boundary conditions.

(b) In one iteration of Jacobi's method, local information propagates only across



(a) First patch **(b)** Second patch

(c) Sparse neighbors **(d)** Withdraw region

(e) Refine withdrawn **(f)** New sparse neighbors

Figure 6.2: Schematic depiction of the stages of the iterative partitioning scheme for an exemplary mesh. Yellow lines denote patch boundaries. (a) Initial creation of a patch where the numbers refer to edge distances to the origin; for visualization purposes only, the triangles which will be removed from the patch in the next step use a smaller font size. (b) Creation of a second patch (blue) starting at one of the unprocessed triangles. (c) Sparse neighbor connection between the two origins of the patches. (d) Withdrawal of sub region in red patch. (e) Refinement of the withdrawn region in the red patch with $d_{max_1} = d_{max_0}/2$. (f) Updated sparse neighbor connections after the red patch is refined.

one edge; using this property the radius of influence is restricted to not exceed the maximal patch radius of $d_{max_0}/2$ by only performing $d_{max_0}/2$ or less iterations.

A linear interpolation between the sparse points on the surface is approximated by using the same boundary conditions (as for the one-dimensional case mentioned above) and starting with the constant extrapolation as an initial guess. (6.1) is not solved until convergence but only a fixed number of iterations of Jacobi's method is

performed.

$$-\nabla^2 \mathbf{u} = 0 \tag{6.1}$$

A finite volume approximation is used to discretize (6.1) on the triangulated mesh by

integrating over the volume $$-\int_{V_i} \nabla^2 \mathbf{u} \, \mathrm{d}V = 0,$$

applying Green's Theorem $$-\int_{\delta V_i} \nabla \mathbf{u} \cdot \mathbf{n_i} \, \mathrm{d}S = 0,$$

summing over the triangle edges $$-\sum_{j=1}^{3} \int_{\delta V_{ij}} \nabla \mathbf{u} \cdot \mathbf{n_{ij}} = 0,$$

using the midpoint rule $$-\sum_{j=1}^{3} L_{E_{ij}} \nabla \mathbf{u} \cdot \mathbf{n_{ij}} \, \mathrm{d}S = 0, \quad \text{and}$$

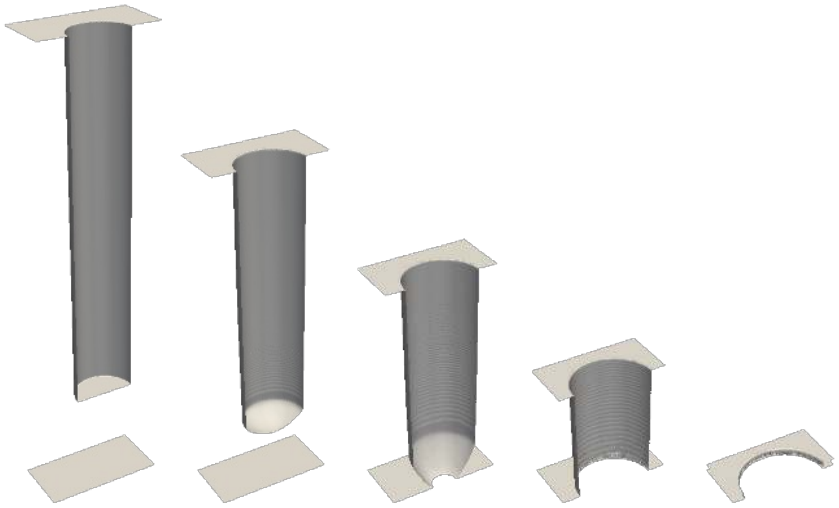
using a central difference between centroids $$\nabla \mathbf{u} \cdot \mathbf{n_{ij}} \approx \frac{\mathbf{u}(x_{n_{ij}}) - \mathbf{u}(x_i)}{\parallel x_{n_{ij}} - x_i \parallel},$$

where $\mathbf{u}$ is the scalar function (in this case the local surface rate), $L_{E_{ij}}$ is the length of the edge shared by triangle i and j, x_i is the centroid of triangle i, and $x_{n_{ij}}$ is the centroid of the triangle connected to triangle i across edge j. This discretization and the boundary conditions described above results in a system of linear equations. The number of unknowns is the number of all centroids minus the size of the sparse set.

6.3 Evaluation and Performance

The simulation platform described in Section 3 is used to implement, validate, and benchmark the approach. A generic etching simulation test case with a single material region is used for the evaluation. The model for the surface velocity is a linear relation to the direct incident flux from a source with a power cosine distribution $\Gamma(\Theta) = cos(\Theta)^{100}$, which is a common choice in *Process TCAD* for the distribution of accelerated ions. The *bottom-up* integration method based on a subdivided icosahedron (cf. Section 4.1) is used to calculate the direct flux rates on the surface. Here, a 5 times subdivided icosahedron is used to limit effects of the integration accuracy. The direct flux rates are normalized to the flux rate on a fully exposed horizontal plane.

The initial geometry (Figure 6.3a) is a cylindrical hole with diameter 1 and depth 6 in a bulk region of thickness 8. Figure 6.3b-6.3e show the intermediate surface positions using the dense centroid-set for surface model evaluation (dense flux evaluation) from time T=0 up to T=8, where the bulk region is completely etched.



(a) T = 0 **(b)** T = 2 **(c)** T = 4 **(d)** T = 6 **(e)** T = 8

Figure 6.3: Cylindrical hole with diameter 1 and depth 6 in a bulk region of thickness 8. Surface evolution during the simulation at times T=[0, 2, 4, 6, 8]. The level-set resolution is 1/64.

To model the refinement condition, for each sparse point i

the maximal normal deviation $$\nu_{max_i} = \max_{\forall k \in N_k} \angle(\mathbf{n}_i, \mathbf{n}_k),$$

the average flux difference $$du_{avg_i} = \frac{1}{n(N_k)} \sum_{\forall k \in N_k} \frac{|u_i - u_k|}{|u_{max} - u_{min}|}, \text{ and}$$

the maximal flux difference $$du_{max_i} = \max_{\forall k \in N_k} \frac{|u_i - u_k|}{|u_{max} - u_{min}|},$$

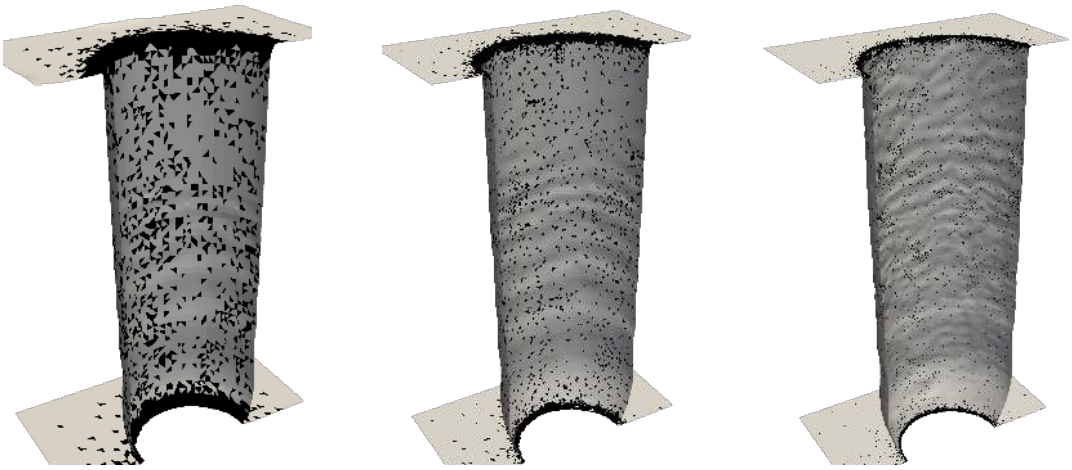
are defined, where N_k is the set of neighboring sparse point indices, and u_{max} and u_{min} are the global maximum and minimum flux value, respectively. A combination of fixed thresholds is used in all of the following results to model the refinement condition in Algorithm 2:

$$RefinementCondition(i) = \begin{cases} true, & \text{if } \nu_{max_i} > \pi/10 \\ true, & \text{if } \frac{du_{avg_i} + du_{max_i}}{2} > 0.2 \\ false, & \text{otherwise} \end{cases} \tag{6.2}$$

Furthermore, $d_{max_0} = 32$ is used in all simulations, which gives a total of 6 iterations (1 initial iteration, and $log2(d_{max_0}) = 5$ refinements), whereas the number of Jacobi iterations is fixed to $d_{max_0}/4 = 8$.

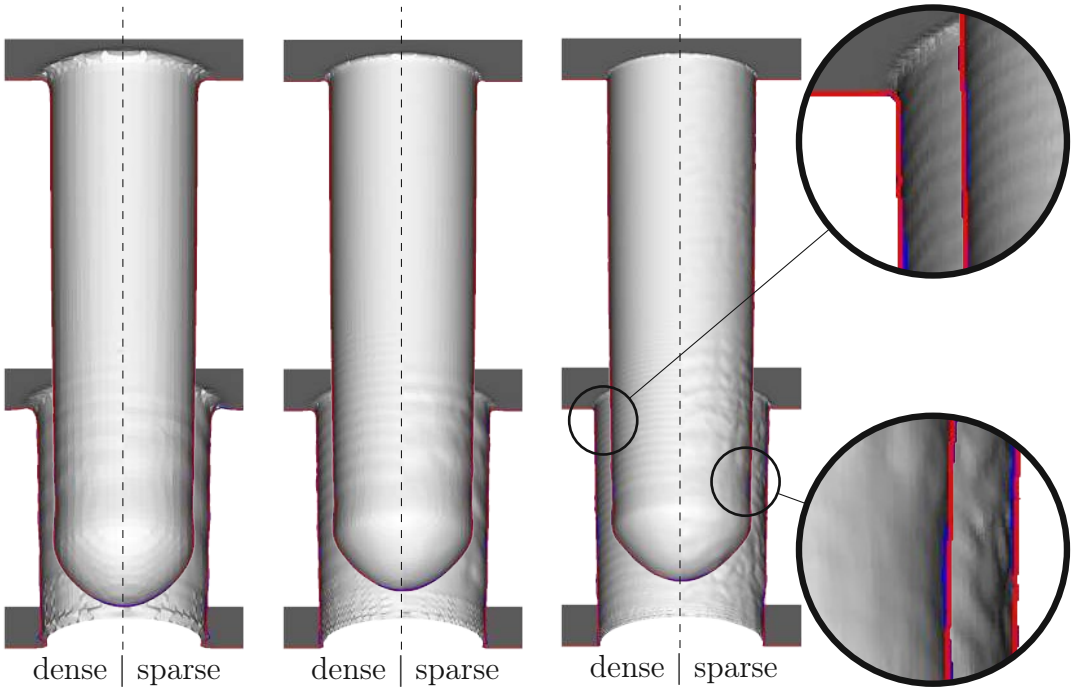
Figure 6.4 illustrates the resulting sparse centroid-set at time T=4.5 for different level-set resolutions. Analogously, Figure C.1 and C.2 in Appendix C illustrate the results for $T = 0$ and $T = 3.0$.

In Figure 6.5, the results between the dense and sparse flux evaluation at times T=3 and T=6 are compared. For a level-set resolution of 64, this corresponds to time step 800 and 1600, respectively. In Figure 6.6, the corresponding maximum



(a) 15412/3264 ≈ 5 **(b)** 56680/5802 ≈ 10 **(c)** 220580/12611 ≈ 17

Figure 6.4: Sparse set of triangles (black; correspond to triangles which are labeled "0" in Figure 6.2) for level-set resolutions 16 (a), 32 (b), and 64 (c) at time T=4.5. The ratios between the total number of triangles and the sparse set of triangles are provided in the subcaptions. These ratios correspond to the *dense/sparse ratios* in Figure 6.7, 6.8, and 6.9.



(a) Resolution 16 **(b)** Resolution 32 **(c)** Resolution 64

Figure 6.5: Comparison of surface positions at times T=[3,6] for all tested resolutions. The surface mesh for the dense and sparse flux evaluation is displayed on the left and right half-space, respectively. Two regions are magnified for resolution 64 where the blue and red line correspond to slices of the sparse and dense evaluation, respectively.

Table 6.1: Level-set resolutions, resulting initial domain resolutions, initial mesh properties, and resulting number of time steps until T=8.

Cells per unit length	Cells vertical	Cells horizontal	Triangles	Time steps
16	128	32x32	17k	540
32	256	64x64	67k	1080
64	512	128x128	262k	2160

deviations throughout the simulations are shown. The maximum deviations peak slightly below 3 level-set cell widths (Δx). For resolution 32, the error stays below Δx. In the upper region of the hole and the top surface, the deviations are small and the sharp edge is conserved.

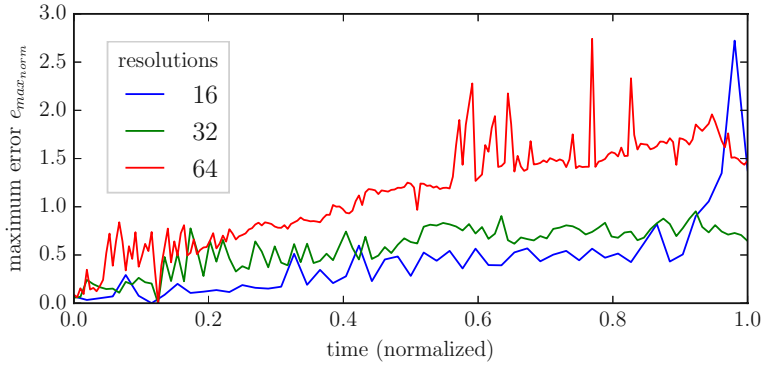


Figure 6.6: Maximum deviation of the surface position throughout the simulations for all three tested resolutions. The deviation is normalized to the resolution, i.e., $e_{max_{norm}} = e_{max}/\Delta x$, where $\Delta x = 1/\text{resolution}$.

The performance of the method is evaluated by tracing the runtime per time step from T=0 to T=8 for three different level-set resolutions summarized in Table 6.1 for the dense and the sparse flux calculation. For each time step, the runtime for the flux evaluation and for the remaining parts (velocity extension, advection, normalization, and mesh extraction; referred to as *other tasks* in the following) is tracked (cf. Figure 6.7-6.9 green and red areas, respectively). The flux integration method for a single point is identical for both cases. The implementation of Algorithm 2 is serial, in contrast to the flux evaluation, which is OpenMP-parallelized in both cases to provide a realistic estimation of the speedups. The serial overhead generated by Algorithm 2 is captured in the runtime of the flux evaluation. All performance measurements were performed on WS1.

Figure 6.7 summarizes the performance differences for resolution 16. The left plot shows the runtime per time step for the dense flux evaluation. The runtime at the beginning of the simulation is ≈ 5.5 seconds per time step. As soon as the hole has reached the bottom of the bulk material, the number of triangles starts to decrease and consequently the runtime per time step drops linearly from T=3.6 to T=8. The ratio between flux evaluation (green) and other tasks (red) is ≈ 20 for the whole simulation, emphasizing the dominance of the computational cost for

the flux evaluation, even for small domain resolutions. The right plot in Figure 6.7 is analogous to the left plot, but for the sparse flux evaluation. A second y-axis on the right is used to plot two additional properties, namely: The ratio of dense to sparse points (dashed line) and the speedup of the flux evaluation (solid line) over the dense flux evaluation. Throughout the simulation, the dense/sparse ratio is between 2.5 and 6 while the speedup is ≈ 2.0.

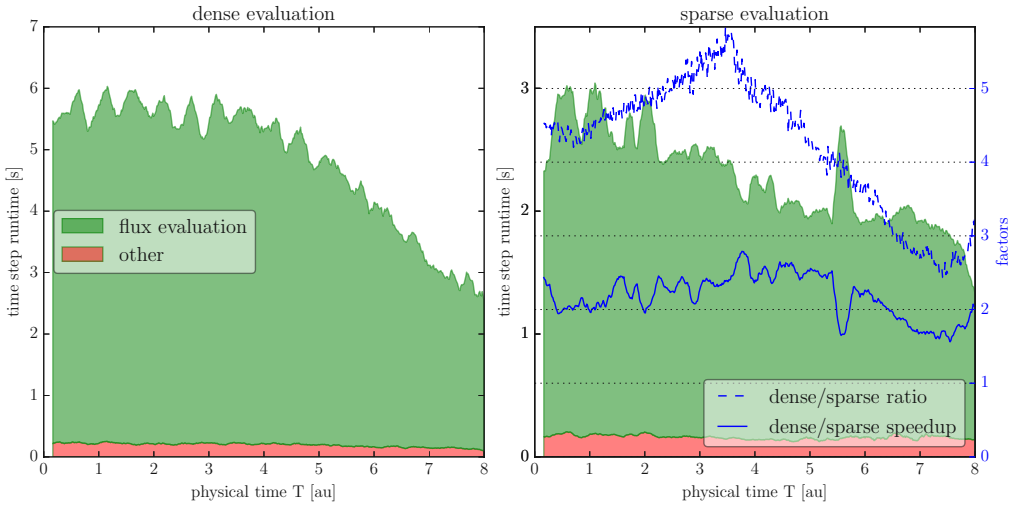


Figure 6.7: Performance results for dense evaluation (left) and sparse evaluation (right) of the direct flux for resolution 16.

Figure 6.8 and 6.9 compare the performance for resolution 32 and 64, respectively. With increasing resolution, the dominance of the flux evaluation in terms of runtime is also increased, leading to a negligible share of runtime for the other tasks in the case of dense flux evaluation. For sparse flux evaluation a dense/sparse ratio of 3 to 14 and 4 to 35 is achieved for resolution 32 and 64, respectively. However, different to resolution 16, the obtained speedups (5 and 8, respectively) are only constant up to T=3.6, where the hole reaches the bottom of the bulk material. From T=3.6 to T=8 the speedups decrease to approximately 2 (following the dense/sparse ratio) keeping the total runtime per time step approximately constant up to T=6.5.

The difference between achieved and potential speedup (i.e., dense/sparse ratio) is higher for large meshes and ranges from ≈ 2 for resolution 16, to ≈ 4 for resolution 64 before the hole reaches the bottom. When approaching T=8, all three tested resolutions converge to a speedup of ≈ 2.

6.4 Summary

A method was presented to reduce the number of necessary evaluation locations for the surface rates to reduce the computational effort with limited impact on the accuracy. A sparse point-set and corresponding neighborhoods are constructed using an iterative partitioning scheme. The surface rates are only evaluated for

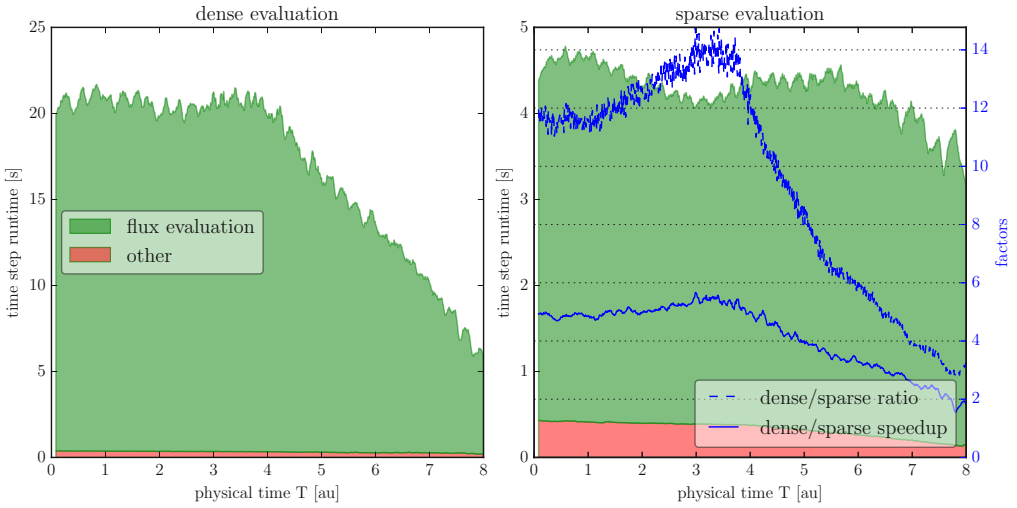


Figure 6.8: Performance results for dense evaluation (left) and sparse evaluation (right) of the direct flux for resolution 32.

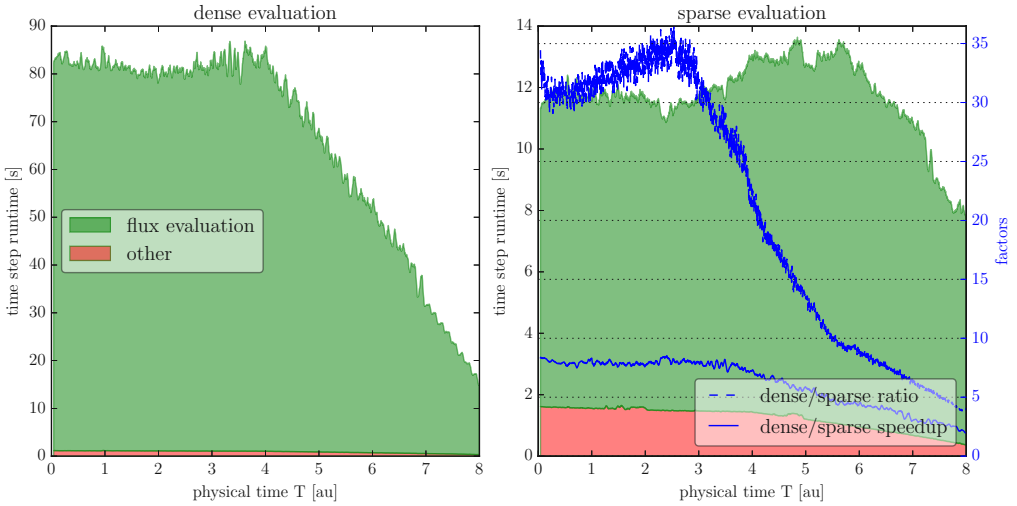


Figure 6.9: Performance results for dense evaluation (left) and sparse evaluation (right) of the direct flux for resolution 64.

this sparse point-set. The variable limits for the allowed distance between sparse locations enable to balance between computational complexity and accuracy in a robust way. A linear interpolation between the sparse points is approximated by diffusing the result of a constant extrapolation in the neighborhoods.

When using a cylindrical hole with a directed vertical source as a generic etching simulation test case, deviations in the surface position are below 3 level-set cells for all tested configurations. The achieved speedups range from 2 for the lowest resolution up to 8 for the highest resolved surface. The speedups are tracked during all time steps of the simulations starting from thick initial geometries to very thin geometries at the end of the simulated physical process.

Chapter 7

Approximation of Flux in High Aspect Ratio Structures

High aspect ratio structures are essential for the fabrication of various semiconductor devices, where the aspect ratio (AR) of the structure is defined as depth/diameter in case of cylinders and as depth/width in case of trenches. One particular example is negative-AND (NAND) flash cell fabrication [71], where three-dimensional multi-layer designs (3D-NAND) involve vertical holes which require aspect ratios above 40. The accurate simulation of the fabrication process for these high aspect ratio structures is challenging, in particular for etching processes: If the re-emission of particles (of the etching species) is neglected, the error in the results for the surface rates increases towards the bottom of the structure.

For a neutral particle species, the flux originating from multiple reflections dominates the surface rates towards the bottom of high aspect ratio structures; surface properties on the sidewalls of a structure which exhibit a high re-emission probability (i.e., low sticking probability) for a neutral particle species emphasize the importance to model a high number of re-emission events. The computational effort for the three-dimensional particle transport calculation increases with the number of considered re-emission events: In a *bottom-up* scheme the number of necessary re-distribution iterations is increased. In a *top-down* Monte Carlo scheme the necessary number of particles is increased in order to obtain an acceptable signal-to-noise ratio at the bottom of a high aspect ratio structure.

In [85] an approach is presented to calculate the neutral flux in long trenches and holes by exploiting symmetry properties of the structures. The three-dimensional problem is reduced to a line integral and the Nyström method [86] is used for discretization. Special numerical treatment is needed to handle singularities during the integration. Spikes and oscillations of the solution near corners of the structure were reported, when the resolution is not refined (compared to the resolution required by the Nyström method) at these critical spots. Assumptions for the transport of the neutral particles are ideal diffuse sources/reflections, a locally constant sticking probability, and molecular flow (ballistic transport without considering inter-particle collisions) of the neutral particles; these assumptions also justify radiosity-based approaches.

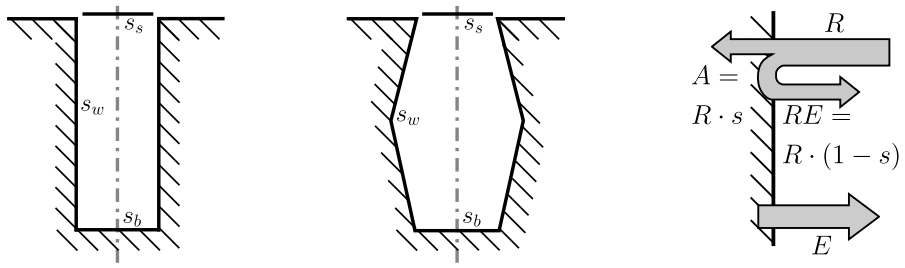
In the following, a radiosity-based one-dimensional approximation for the surface

rates of a neutral particle species is presented for convex holes and trenches [87]. The approximation is eligible to replace the three-dimensional particle transport calculation for neutral particle species. It is applicable to simulations where the three-dimensional geometry can be approximated with a convex rotationally symmetric hole or convex symmetric trenches. Although these constraints seem restrictive, the approach can be an attractive choice in modern *Process TCAD*, in particular for memory devices where symmetric high aspect ratio structures are utilized to increase the integration density.

First, the details of the approach are presented. Then, the utilized view factors are introduced including a novel combination of analytical view factors to obtain an analytical view factor for coaxial cones. Then, the approximation is validated using the three-dimensional *top-down* particle transport implementation of *ViennaTS*[14]. Finally, the capabilities of the presented approach are demonstrated.

7.1 One-Dimensional Radiosity-Based Particle Transport

For cylindrical holes, the simulation domain is a rotationally symmetric closed convex surface. For trenches, the simulation domain is a trench with a closed convex symmetric cross section. The neutral flux source is modeled by closing the structures at the top. This leads to a disk-shaped source for holes and a strip-shaped source for trenches. Figure 7.1a and Figure 7.1b illustrate the cross sections of domains with vertical walls and with a kink at one half of the depth, respectively.



(a) Vertical domain **(b)** Kinked domain **(c)** Surface model

Figure 7.1: Cross sections of simulation domains with vertical walls (a) and with a kink at one half of the depth (b). s_s, s_w, and s_b designate the sticking probabilities for the source, the wall, and bottom region, respectively. (c) illustrates the surface model showing the relation between the received flux R, the adsorbed flux A, and the re-emitted flux RE; source areas emit a flux E independent of the received flux R.

The surface adsorption is modeled using a locally constant sticking probability s. The received flux R is split according to s into an adsorbed flux A and a re-emitted flux RE as depicted in Figure 7.1c. Source areas additionally emit a flux E independent of R.

The discrete form of the radiosity equation (2.15) is used. For a surface element i the received flux R_i is

$$R_i = \sum_j (E_j F_{ji}) + \sum_j \left((1 - s_j) R_j F_{ji} \right), \tag{7.1}$$

where E_j is the self-emitted energy, s_j is the sticking probability, and F_{ji} is the view factor (proportion of the radiated energy, which leaves element j and is received by element i).

The linear system of equations is obtained by rewriting (7.1) in matrix notation

$$\vec{R} = \boldsymbol{F}^T \cdot \vec{E} + \operatorname{diag}(1 - \vec{s})\, \boldsymbol{F}^T \cdot \vec{R}\,, \tag{7.2}$$

and transformation into the standard form

$$\left(\boldsymbol{I} - \operatorname{diag}(1 - \vec{s})\, \boldsymbol{F}^T \right) \cdot \vec{R} = \boldsymbol{F}^T \cdot \vec{E}\,, \tag{7.3}$$

with the vector of emitted flux $\vec{E}$, a vector of sticking probabilities $\vec{s}$, and a matrix of view factors $\boldsymbol{F}$ (where F_{ij} corresponds to the view factor $i \to j$).

The solution of the diagonally-dominant linear system of equations (7.3) is approximated using the Jacobi method. The number of performed Jacobi-iterations corresponds to the considered number of re-emissions of each element to all other elements. The adsorbed flux A is related to R by the corresponding sticking probability s of the element

$$A_i = R_i s_i\,. \tag{7.4}$$

The relation $\| \vec{A} \| - \| \vec{E} \| = 0$, which holds for closed surfaces, can be used to test the implementation and to define a stopping criterion for the Jacobi iterations.

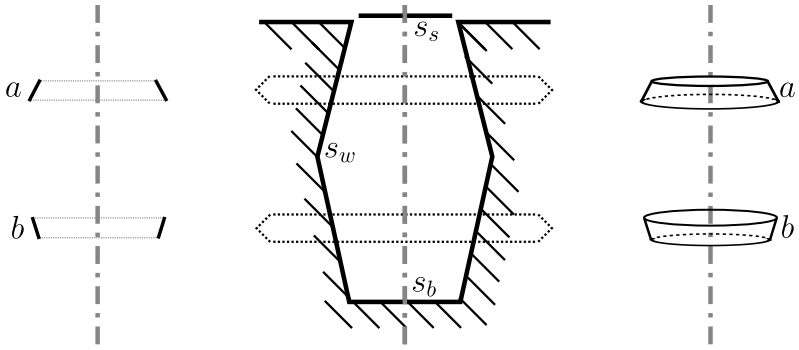
7.2 View Factors

The approach presented here is based on the discretization of the surface into discrete surface elements along the structure's line of symmetry. Figure 7.2 shows the cross section of a convex structure and the shape of the resulting surface elements. Two vertical ranges are indicated in Figure 7.2b and the resulting surface elements a and b are shown for a trench (Figure 7.2a) and a hole (Figure 7.2c). The elements are formed from two strips for the trench and take the form of a sliced cone for the hole.

To assemble the matrix $\mathbf{F}$ the view factors between all possible pairs of surface elements are required.

7.2.1 Trench View Factors

The view factor between two segments of a symmetric convex trench with a constant cross section, as depicted in Figure 7.2a, is derived using the crossed-strings method [88]. This method computes the view factor between two surfaces with a constant cross section and infinite length utilizing a two-dimensional re-formulation



(a) Trench **(b)** Domain **(c)** Hole

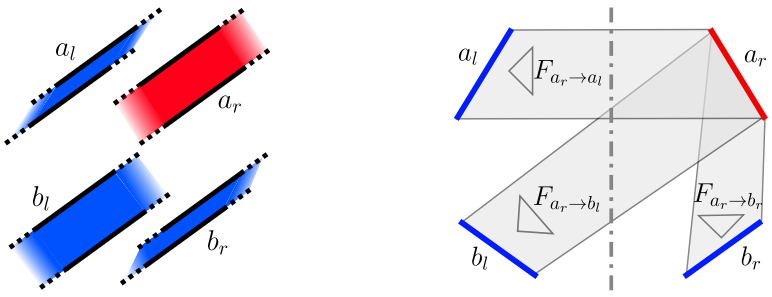
Figure 7.2: Two surface elements, which result when discretizing the domain (b) are displayed: (a) is the side view of two surface elements a and b, which result from a trench discretization and (c) is the isometric view of two surface elements a and b, which result from a hole discretization.

of the problem. For two mutually completely visible strips of infinite length the view factor is [88]

$$F_{1\to 2} = \frac{(d_1 + d_2) - (s_1 + s_2)}{2 \cdot a_1}, \tag{7.5}$$

where d_1 and d_2 denote the lengths of the diagonals, when connecting the cross section of the two strips to form a convex quadrilateral, s_1 and s_2 denote the lengths of the sides of that quadrilateral which connects the strips, and a_1 denotes the length of the side of the quadrilateral which represents the emitting strip.

Figure 7.3a is an isometric view of the four strips from Figure 7.2a. The view factors from the top right strip a_r towards the other three strips is visualized in Figure 7.3b.



(a) Isometric view **(b)** Side view

Figure 7.3: Isometric (a) and side view (b) on the four infinite strips which correspond to the surface elements a and b from Figure 7.2a. In (b) the view factors from the top right strip a_r towards the other three strips are visualized.

The view factor between the two segments a and b is

$$F_{a\to b} = F_{a_r\to b_r} + F_{a_r\to b_l}, \tag{7.6}$$

where the subscripts denote the side of the strip according to Figure 7.3b; a_l can be neglected, as the cross section is symmetric. The view factor from an element to itself is

$$F_{a\to a} = F_{a_r\to a_l}, \tag{7.7}$$

where again the other direction can be neglected due to symmetry. Equation (7.5) is used to compute the view factors between individual strips in (7.6) and (7.7).

7.2.2 Hole View Factors

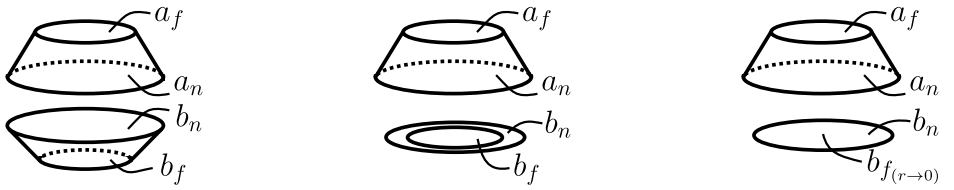
A general formulation to compute the view factors between two segments of a rotationally symmetric convex hole (cf. Figure 7.2c) is derived. It is based on the view factor between two coaxial disks of nonequal radii r_1 and r_2 at a distance z defined by

$$F_{1\to 2} = \frac{1}{2}\left(X - \sqrt{X^2 - 4(R_1/R_2)^2}\right), \tag{7.8}$$

where $R_i = r_i/z$ and $X = 1+(1+R_2^2)/R_1^2$ [89]. Using this relation and the reciprocity theorem of view factors

$$S_1 \cdot F_{1\to 2} = S_2 \cdot F_{2\to 1}, \tag{7.9}$$

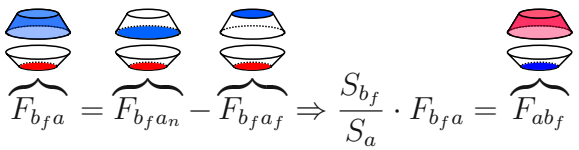
where S is the element area, a general formulation for the view factor between the inner wall surfaces of two coaxial cone-like segments (whose surfaces are mutually completely visible) is obtained. Figure 7.4a shows two segments a and b in such a configuration and denotes the four coaxial disks which represent the apertures of the two elements.



(a) Cone/cone **(b)** Cone/annulus **(c)** Cone/disk

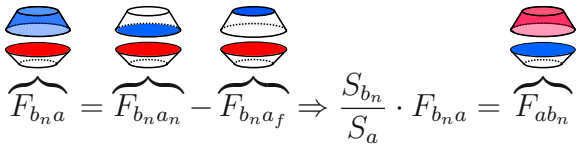
Figure 7.4: Three possible pairs of segments as they result from discretizing the hole. For each pair, the *near* apertures a_n and b_n, and the *far* apertures a_f and b_f are denoted: (a) two cone-like segments, (b) cone and annulus, and (c) cone and disk. The *far* aperture is treated as an infinitely small element.

The final goal to compute the view factor between two elements a and b is divided into multiple inexpensive view factor computations between coaxial disks. First, the difference of the view factors from b_f towards the two disks of a is computed, and the reciprocity theorem (7.9) is applied to obtain F_{ab_f} (red indicates *sending* and blue *receiving* areas).



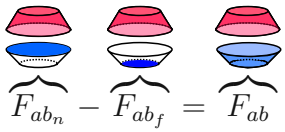
(7.10)

The same is done for b_n to obtain F_{ab_n}.



$$F_{b_n a} = F_{b_n a_n} - F_{b_n a_f} \Rightarrow \frac{S_{b_n}}{S_a} \cdot F_{b_n a} = F_{a b_n} \tag{7.11}$$

Finally F_{ab} is obtained by subtracting F_{ab_f} from F_{ab_n}.



$$F_{ab_n} - F_{ab_f} = F_{ab} \tag{7.12}$$

The view factor of an element to itself F_{aa} is computed by subtracting the flux leaving through the two apertures from unity.

$$F_{aa} = 1 - F_{aa_n} - F_{aa_f} \tag{7.13}$$

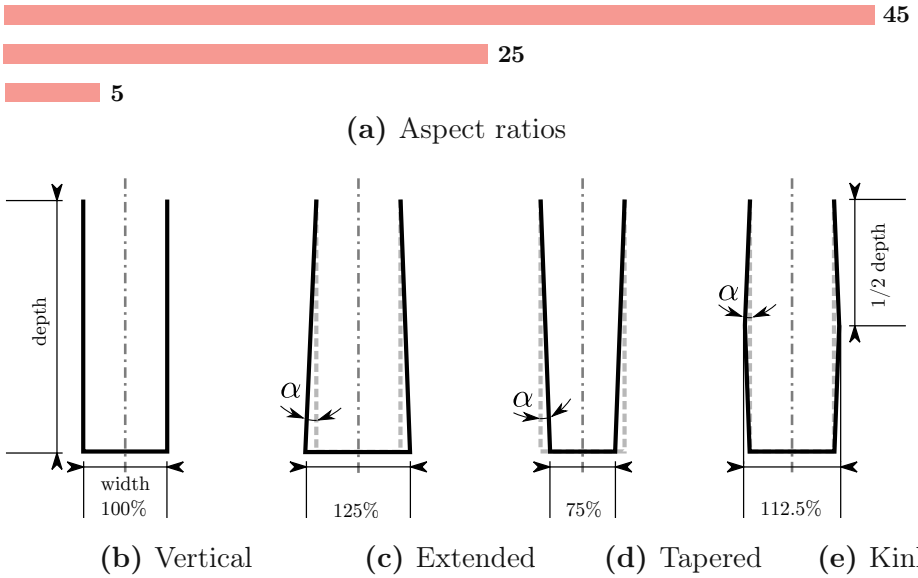
If an element is an annulus or a disk (see Figure 7.4b and Figure 7.4c, respectively), the general formulation still applies. For a disk, the *far* aperture is treated as an infinitely small element.

7.3 Validation and Results

The sticking probabilities for the wall and the bottom of the structures are selected to represent a reasonable approximation to the prevalent conditions for the neutral particles in an ion-enhanced chemical etching [90] (IECE) environment. A sticking probability $s_s = 1$ is used for source areas which do not have any reflections originating from these artificial areas; the bottom is modeled as a fully adsorbing area with a sticking probability $s_b = 1$. A constant sticking probability s_w is used for the walls of the structures.

The results for cylindrical holes with different aspect ratios (5 to 45; cf. Figure 7.5a), sticking probabilities s_w (0.02 to 0.2), and geometries (cf. Figure 7.5b-7.5e) are compared with the reference results obtained using *ViennaTS* [14], which uses a three-dimensional *top-down* Monte Carlo approach for the particle transport. The results show good agreement (cf. Figure 7.6) aside from the deviation at the wall/bottom interface, caused by the discretization which is used in the reference simulation. Figure D.1 in Appendix D plots the flux distributions for a hole and a trench of aspect ratio 25 along the wall and at the bottom for sticking probabilities $s_w = 0.2$ and $s_w = 0.01$: All results are in agreement with the result obtained by the reference simulation.

The neutral particle flux at the bottom of a structure is an important parameter during an IECE process, as it determines the etch rate of the process [85]. Figure 7.7 shows results obtained with the presented approach for the flux at the bottom center of a hole and a trench structure. The sticking probability of the bottom is set to $s_b = 1$ (cf. Figure 7.7a, 7.7b) and $s_b = s_w$ (cf. Figure 7.7c, 7.7d). The total flux (solid lines) and the flux originating from re-emission (indirect flux, dashed lines)



(a) Aspect ratios **(b)** Vertical **(c)** Extended **(d)** Tapered **(e)** Kinked

Figure 7.5: (a) True to scale aspect ratios from 5 to 45. (b)-(e) Cross sections of the geometric variations of the wall for holes and trenches (shown for AR=3); the resulting angle α, which is identical for all three variations, is depicted.

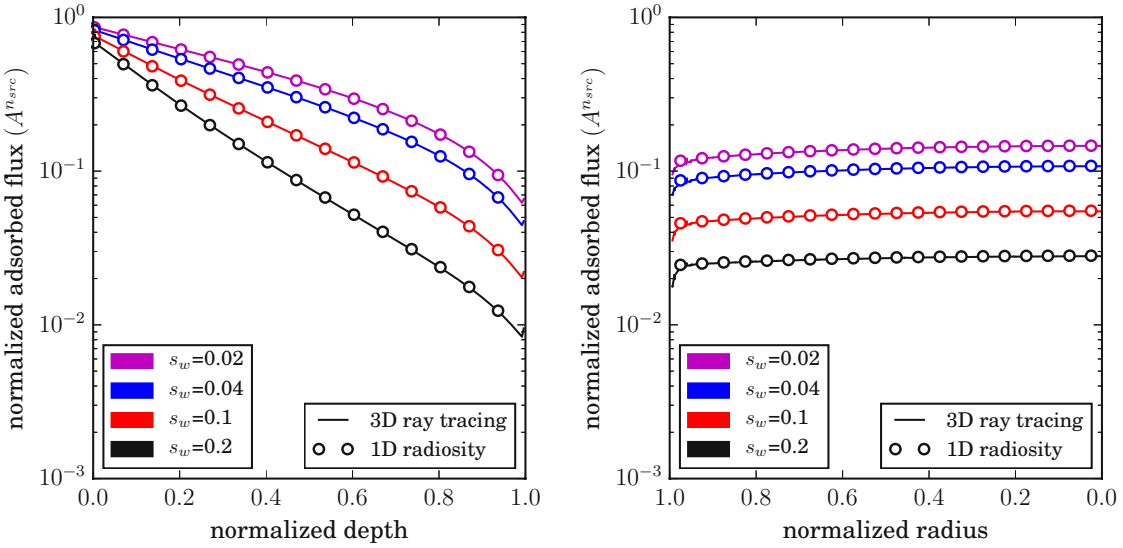
is plotted for aspect ratios between 0.1 and 50, and different sticking probabilities s_w. The results reveal the effect of a high sticking probability at the bottom for high aspect ratio structures: The bottom adsorbs more particles, which leads to a higher contribution of the direct flux. For instance, for a hole with aspect ratio 50 and $s_w = 0.02$, the ratio indirect/total flux is 0.33 and 0.8 for $s_b = 1$ and $s_b = 0.02$, respectively.

7.4 Summary

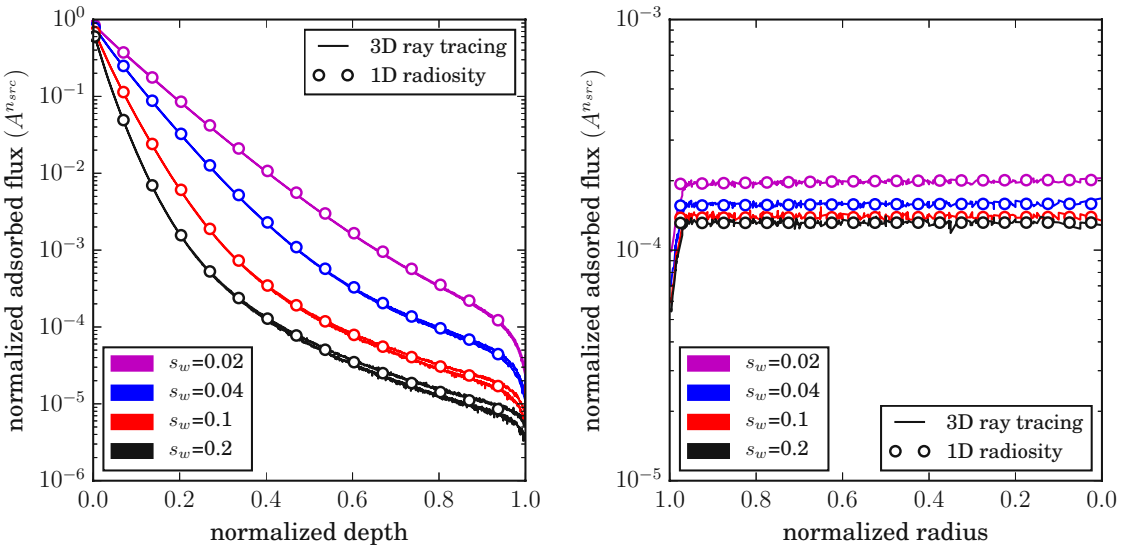
A computationally inexpensive radiosity-based approximation of the local neutral flux for three-dimensional plasma etching simulations of high aspect ratio holes and trenches was presented. All relevant view factors for holes are computed by establishing an inexpensive general formulation for the view factor between coaxial cone-like segments.

It can be used as a drop-in replacement for the neutral flux computation during three-dimensional IECE simulations of high aspect ratio structures offering an underlying symmetry, as shown here for holes and trenches, to significantly reduce simulation times in practical simulation cases — or as a stand-alone tool which provides fast results for exploratory investigations.

Comparing the results for various convex configurations using a rigorous three-dimensional Monte Carlo ray tracing simulation shows good agreement.

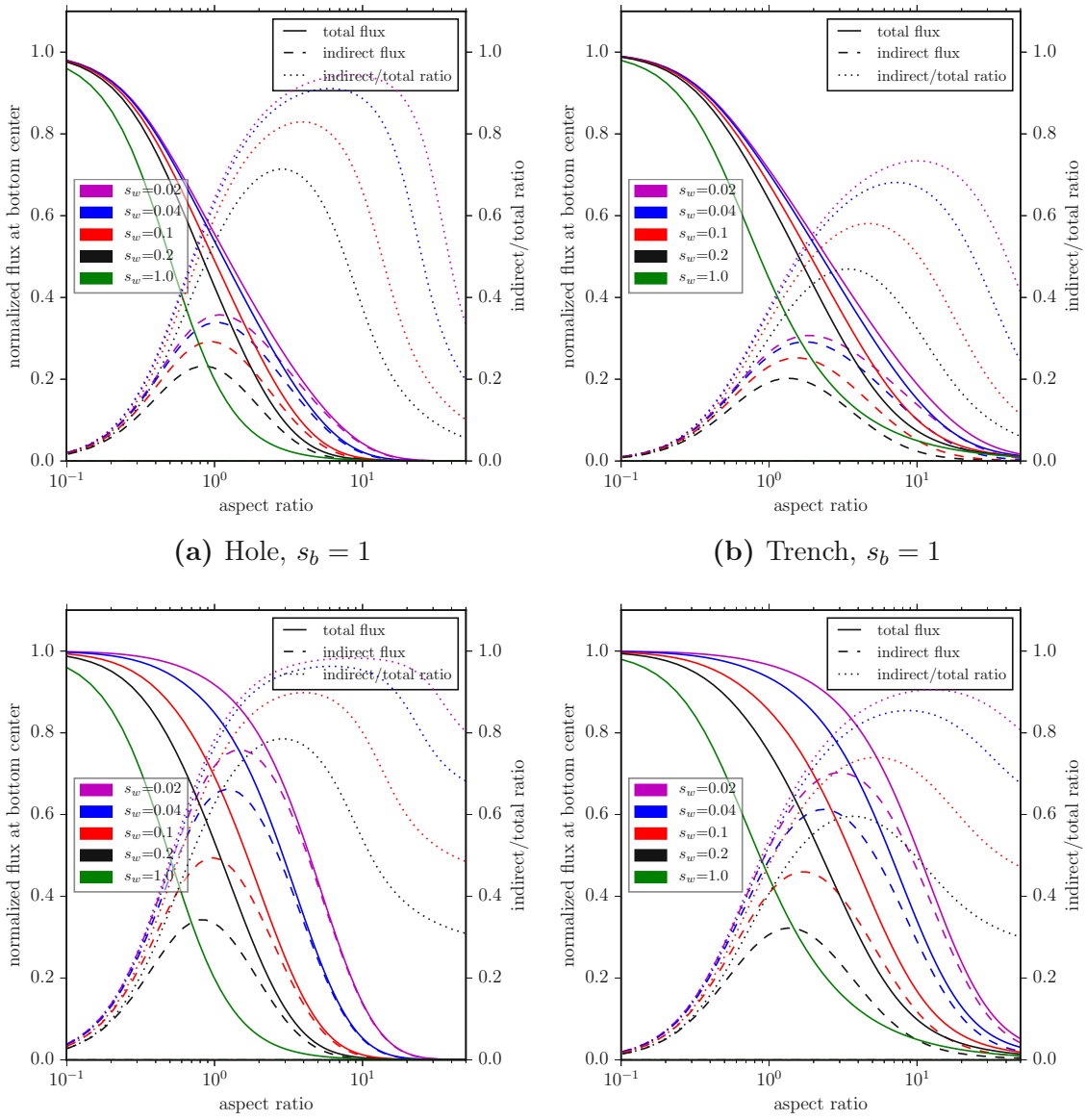


(a) Cylinder wall, AR=5 **(b)** Cylinder bottom, AR=5



(c) Cylinder wall, AR=45 **(d)** Cylinder bottom, AR=45

Figure 7.6: Normalized flux distributions along the wall and at the bottom for cylinders of aspect ratio 5 (a)(b) and aspect ratio 45 (c)(d). The one-dimensional radiosity approach (circles) is compared to the results of a three-dimensional ray tracing simulator (lines). The sticking probability of the wall s_w is varied between 0.02 and 0.2. The deviations between ray tracing and radiosity towards the wall-bottom interface are due to the resolution of the ray tracing simulator. In (c) the ray tracing results are plotted using the minimum and maximum along the cylinder radius, particularly visible for $s_w = 0.2$.



(a) Hole, $s_b = 1$ **(b)** Trench, $s_b = 1$

(c) Hole, $s_b = s_w$ **(d)** Trench, $s_b = s_w$

Figure 7.7: Total flux (solid) and indirect flux (dashed) at the bottom center of a vertical hole and trench structure for various aspect ratios (0.1 to 50) and different sticking probabilities (s_w) of the wall (0.02 to 1). The ratio between indirect and total flux (dotted) is plotted additionally using the right y-axis. (a)-(b) Result for a sticking probability $s_b = 1$ at the bottom. (c)-(d) Result for a sticking probability $s_b = s_w$ at the bottom.

Chapter 8

Combining Acceleration Techniques for Direct Flux

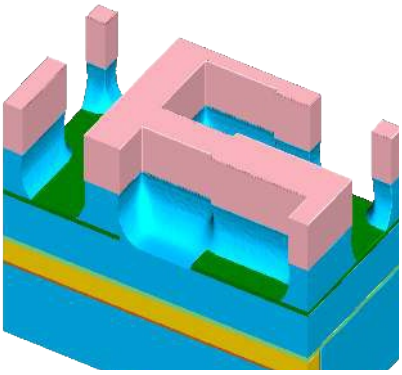
The approaches to accelerate the flux calculation presented in Chapters 4, 5, and 6 are applied to the etching simulation of a dielectric layer introduced in Section 1.2.4 to showcase the advantage of their combined use. The surface velocity V_n is defined as a linear relation to the direct flux F_{direct} from a power cosine source (cf. Figure 4.3) with exponent $n = 100$.

$$V_n(\mathbf{x}) = \begin{cases} -0.01F_{direct}(\mathbf{x}), & \text{if } M(\mathbf{x}) = M_{photoresist} \\ -0.02F_{direct}(\mathbf{x}), & \text{if } M(\mathbf{x}) = M_{barrier} \vee M_{etchstop} \\ -0.1F_{direct}(\mathbf{x}), & \text{if } M(\mathbf{x}) = M_{metal} \vee M_{seed} \\ -1.0F_{direct}(\mathbf{x}), & \text{if } M(\mathbf{x}) = M_{dielectric} \end{cases} \tag{8.1}$$

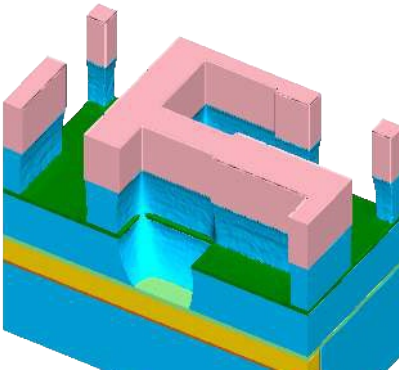
The etching is simulated until $T = 1.5$, where the etchstop embedded in the dielectric and the diffusion barrier is reached at $T = 0.5$ and $T = 1.0$, respectively; Figure 8.1a-8.1c show the corresponding cross sections of the simulation domain.

All benchmarks in this chapter are performed on WS2. As a frame of reference, for resolution 1/64, the runtimes of the main computational tasks with a 4 times subdivided icosahedron for spherical sampling (cf. Section 4.1) and implicit ray tracing (using *OpenVDB*) are shown in Figure 8.1d. Analogously, Figure 8.1e shows the runtimes for the same setup, when applying the presented acceleration schemes, namely (a) explicit ray tracing on a temporary explicit mesh (using *Embree*, cf. Section 5.3), (b) an adaptive sampling of the visibility directions with one level of refinement (i.e., *minlevel*= 3 and *maxlevel*= 4, cf. Section 4.3), and (c) a sparse evaluation of the surface rates using a maximum edge distance of 16 (i.e., $d_{max0} = 16$, cf. Section 6.1) and Equation (6.2) as refinement condition. The overall runtime for a time step is about 70 seconds at the beginning of the simulation and about 110 seconds for $T = 1.5$. When applying all presented approaches combined the overall runtime is reduced to about 5 and 8 seconds, respectively. This corresponds to an overall speedup of about 14.

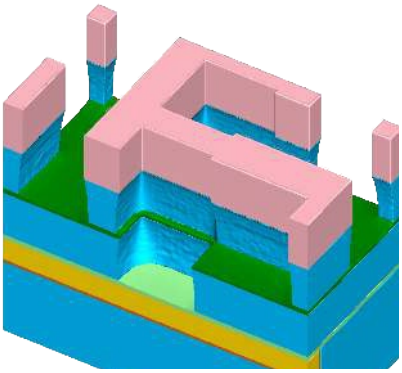
The speedup for the calculation of the surface rates is shown in Figure 8.2 for various additional combinations of the presented acceleration approaches. The speedups are normalized to runtimes based only on an explicit ray tracing approach. The con-



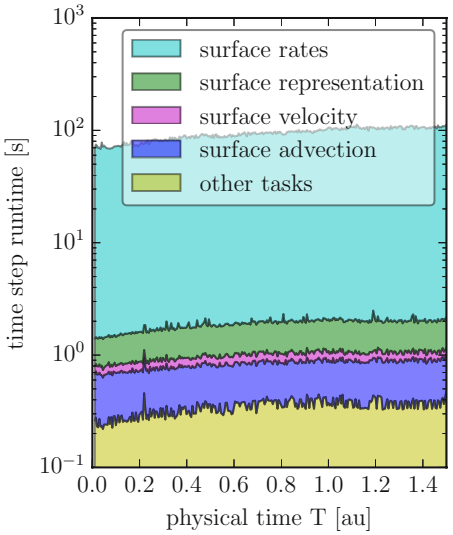
(a) $T = 0.5$, shortly after the etch stop layer was reached



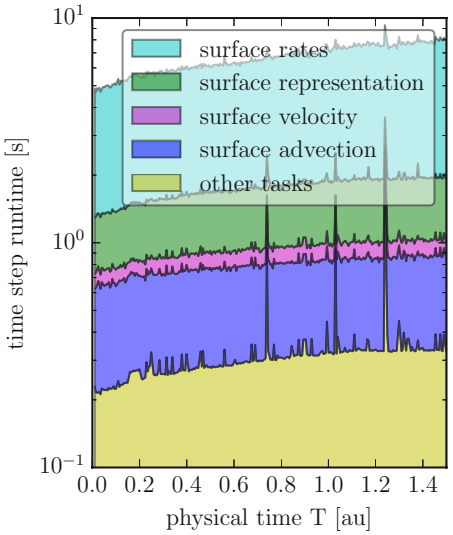
(b) $T = 1.0$, shortly after the diffusion barrier was reached



(c) $T = 1.5$, final topography



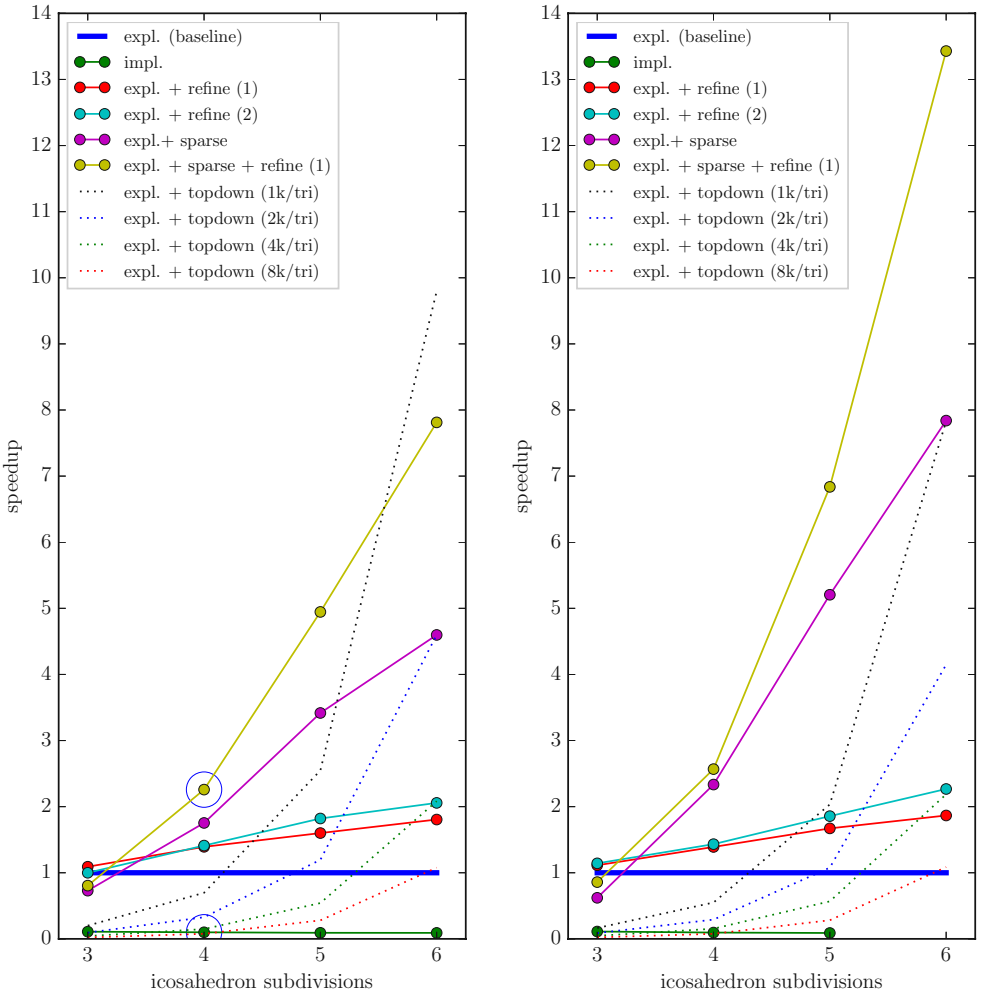
(d) Implicit tracing



(e) All methods combined

Figure 8.1: Results of a feature-scale etching simulation of the dielectric layer in a "self-aligned dual-damascene process" (cf. Figure 1.1h). The lateral dimensions of the domain are 192×192 grid cells. (a)-(c) Cross sections of the domain for $T = [0.5, 1.0, 1.5]$. (d) Runtimes demanded with implicit tracing. (e) Runtimes using all presented acceleration approaches combined.

figurations corresponding to the result shown in Figure 8.1d and 8.1e are additionally marked with a circle. Figure 8.2 confirms the speedups estimated in the individual chapters presenting the acceleration approaches: The explicit ray tracing (which is



(a) Resolution 1/64 **(b)** Resolution 1/128

Figure 8.2: Average speedup for the surface rate calculation for various combinations of the presented acceleration approaches normalized to the runtime with solely the explicit tracing approach (*expl.*). (a) and (b) show the results for resolution 1/64 and 1/128, where the configurations are otherwise identical besides the d_{max_0} setting for the *sparse* evaluation of 16 and 32, respectively. The results corresponding to Figure 8.1d and 8.1e are additionally marked with a circle in (a). The speedup with a *top-down* Monte Carlo approach for the direct flux is plotted in dashed lines where each line represents a constant number of launched rays, i.e., *1k/tri* corresponds to 1000 triangles launched per triangle on an equivalent horizontal surface.

used as a baseline, thick blue line) provides a speedup of about 9 compared to the implicit tracing (green line). The adaptive visibility sampling provides a speedup of 2 for a 6 times subdivided icosahedron; for lower subdivision levels, the speedup decreases to only a marginal speedup for 3 subdivisions. The sparse evaluation of the surface rates provides a speedup of 1.5 and 2 for 4 subdivisions up to 4.5 and 8 for 6 subdivisions. The combined speedup can be approximated by the product of the individual speedups. The speedup for the combined approaches is above 14 for 4 or more subdivisions.

Figure 8.2 additionally plots the speedups when using explicit tracing combined with a *top-down* Monte Carlo approach for the direct flux calculation. The speedups are plotted for four different numbers of launched rays per triangle on a equivalent horizontal surface, i.e., for resolution 1/64 the launched number of rays is $192 \times 192 \cdot 2 \cdot 1000 \approx 73$Mrays, where 2 triangles are assumed per grid cell and 1000 rays are launched per triangle (*1k/tri*).

Figure 8.2 also allows to relate the runtime between *top-down* and *bottom-up* approaches for the direct flux calculation, e.g., for resolution 1/64 the runtime for a *top-down* scheme using *1k/tri* is about a factor of 0.7 slower than a sampling using a 4 times subdivided icosahedron. This relation can merely be used as a rough estimate as it strongly depends on the geometry, if the number of launched rays is related to the number of equivalent triangles of a horizontal surface.

Chapter 9

Summary and Outlook

The state of the art of numerical methods for topography simulation for *Process TCAD* was presented and discussed with a focus on computational aspects. In particular, different approaches for particle transport and surface advection were discussed. The level-set method was identified as predominant choice for surface advection for three-dimensional process simulation. Thus, from an implementation point of view, the starting position for any particle transport/flux calculation approach is a set of level-sets representing the material regions in the simulation domain. All common approaches for the particle transport, which all assume ballistic transport in the feature-scale region, rely on a vast number of ray-surface intersection tests to either perform visibility tests or to simulate the trajectories of particles in the simulation domain. This emphasizes that an indispensable requirement for a high performance particle transport/flux calculation approach is access to a highly efficient ray casting back end.

With these two requirements in mind, a simulation framework was developed to explore novel approaches for particle transport/flux calculations. The framework combines open-source third-party libraries for sparse volumetric data and ray tracing to store and advect the level-sets and to perform the ray-surface intersection tests. Using this framework, different approaches to reduce the computational workload of the particle transport/flux calculation were investigated.

The accuracy requirements for the ray-surface intersection tests was put in relation to the accuracy obtained with single-precision arithmetics for the ray casting: It is admissible to utilize single-precision arithmetics for the ray-surface intersection tests in practical process simulation scenarios. This is a fundamental finding as all particle transport methods benefit from the improved underlying performance.

As the surface advection is predominantly level-set-based, no explicit representation of the surfaces is available. Using two highly optimized open-source libraries (from the field of computer graphics) for ray casting on implicit (*OpenVDB*) and explicit (*Embree*) surfaces, it is shown that the overhead introduced by an extraction of a temporary polygonal mesh (in each time step of the simulation) is by far compensated by the performance gain obtained from the ray casting against the explicit surface.

To reduce the runtime of a *bottom-up* direct flux calculation two approaches were pursued: Firstly, the number of necessary visibility sampling directions (per

integration point) is reduced by adaptively refining the sampling only around the boundary of the aperture regions using a hierarchical subdivision of the spherical directions. Secondly, the number of integration points on the surface is locally reduced (according to a freely definable application-specific condition) using an iterative partitioning scheme on the extracted polygonal surface mesh. Both approaches reduce the runtime of the direct flux calculation significantly. The accuracy of the result is not influenced by the first approach and the influence of the second approach is reasonably small.

All approaches are applicable for arbitrary three-dimensional geometries and were applied in conjunction to an etching simulation of a multi-material stack. The resulting overall simulation speedup is above 14 for a wide range of settings.

A further utilization of the application-specific refinement condition for the iterative partitioning has potential to increase the speedup – for example by introducing a material dependence, e.g., exploiting the fact that the demands for accuracy can differ greatly from material to material, or by introducing a dependence on the position in the domain, e.g., the accuracy demands can be tailored to simulations of high aspect ratio structures.

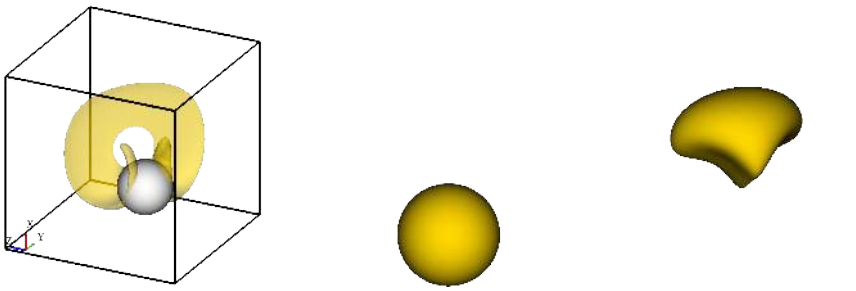
Introducing a dependence on the emission characteristics of the source for the adaptive visibility sampling and the subsequent integration potentially increases viability of this approach further. This is especially auspicious with regard to the integration accuracy for very directed sources.

Appendix A

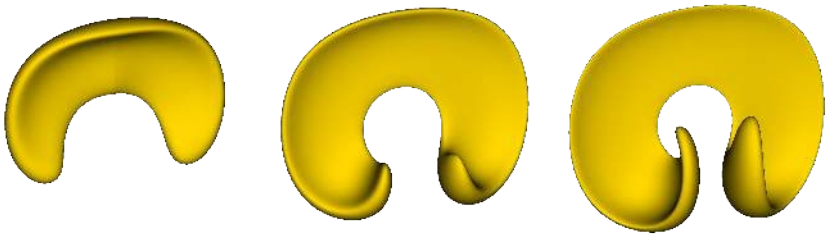
Supplementary Material Chapter 3

In the following, the results for the test cases in Chapter 3 are provided for various resolutions.

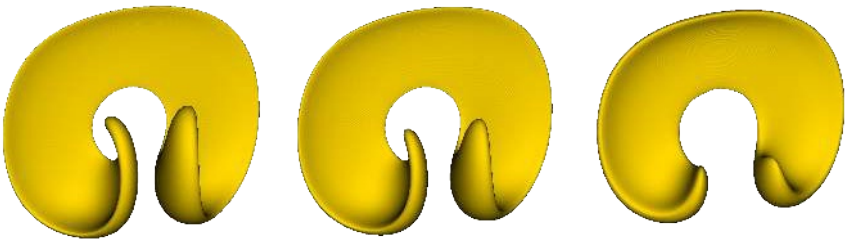
A.1 Enright Test



(a) Unit Domain **(b)** $T = 0.0$ **(c)** $T = 0.3$



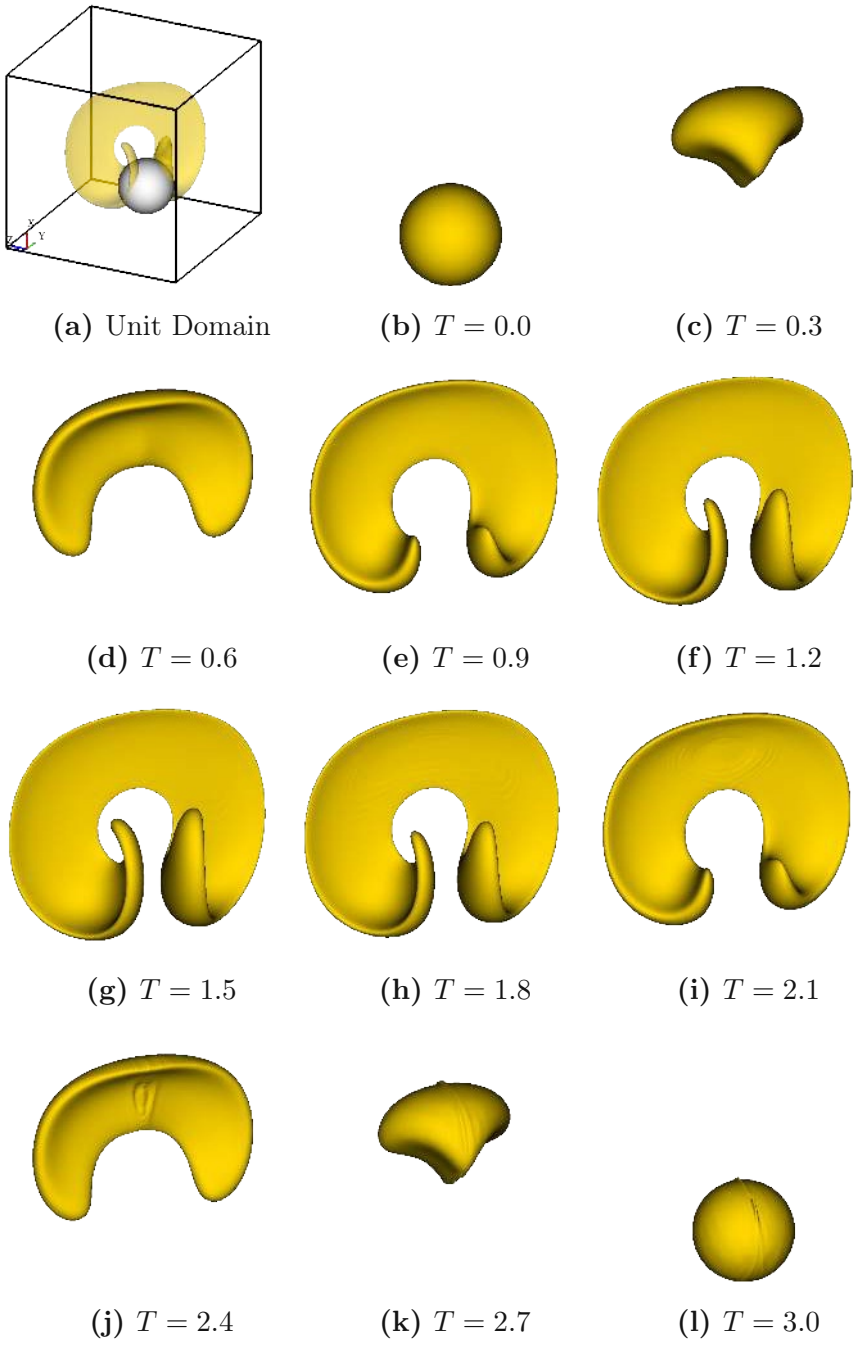
(d) $T = 0.6$ **(e)** $T = 0.9$ **(f)** $T = 1.2$



(g) $T = 1.5$ **(h)** $T = 1.8$ **(i)** $T = 2.1$

(j) $T = 2.4$ **(k)** $T = 2.7$ **(l)** $T = 3.0$

Figure A.1: Results for the *Enright test* for resolution 1024^3. The domain is shown in (a); (b)-(l) show the result from $T = 0.0$ to $T = 3.0$ with a stepsize 0.3. The same illustration for resolution 512^3, 256^3, and 128^3 is provided in Figures A.2, A.3, and A.4.



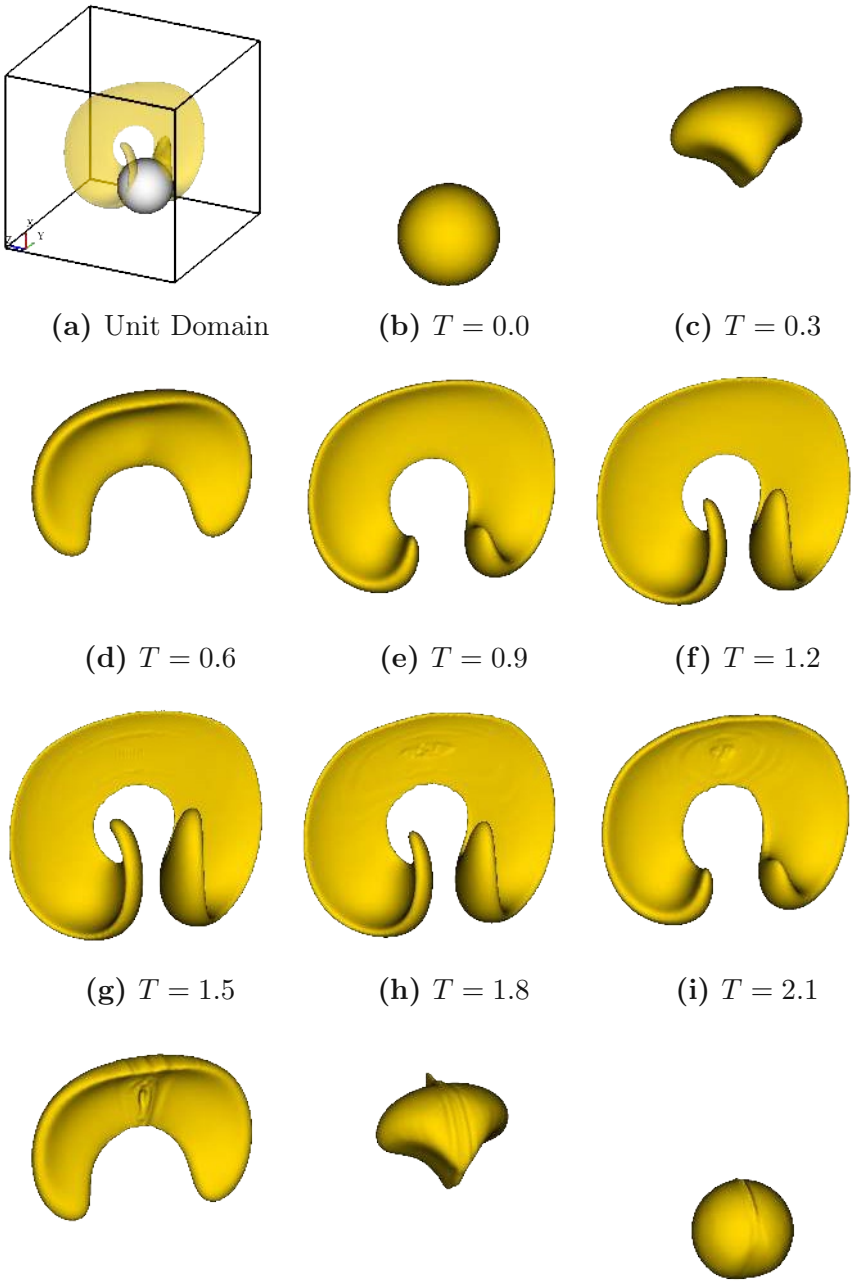
(a) Unit Domain **(b)** $T = 0.0$ **(c)** $T = 0.3$

(d) $T = 0.6$ **(e)** $T = 0.9$ **(f)** $T = 1.2$

(g) $T = 1.5$ **(h)** $T = 1.8$ **(i)** $T = 2.1$

(j) $T = 2.4$ **(k)** $T = 2.7$ **(l)** $T = 3.0$

Figure A.2: Results for the *Enright test* for resolution 512^3.



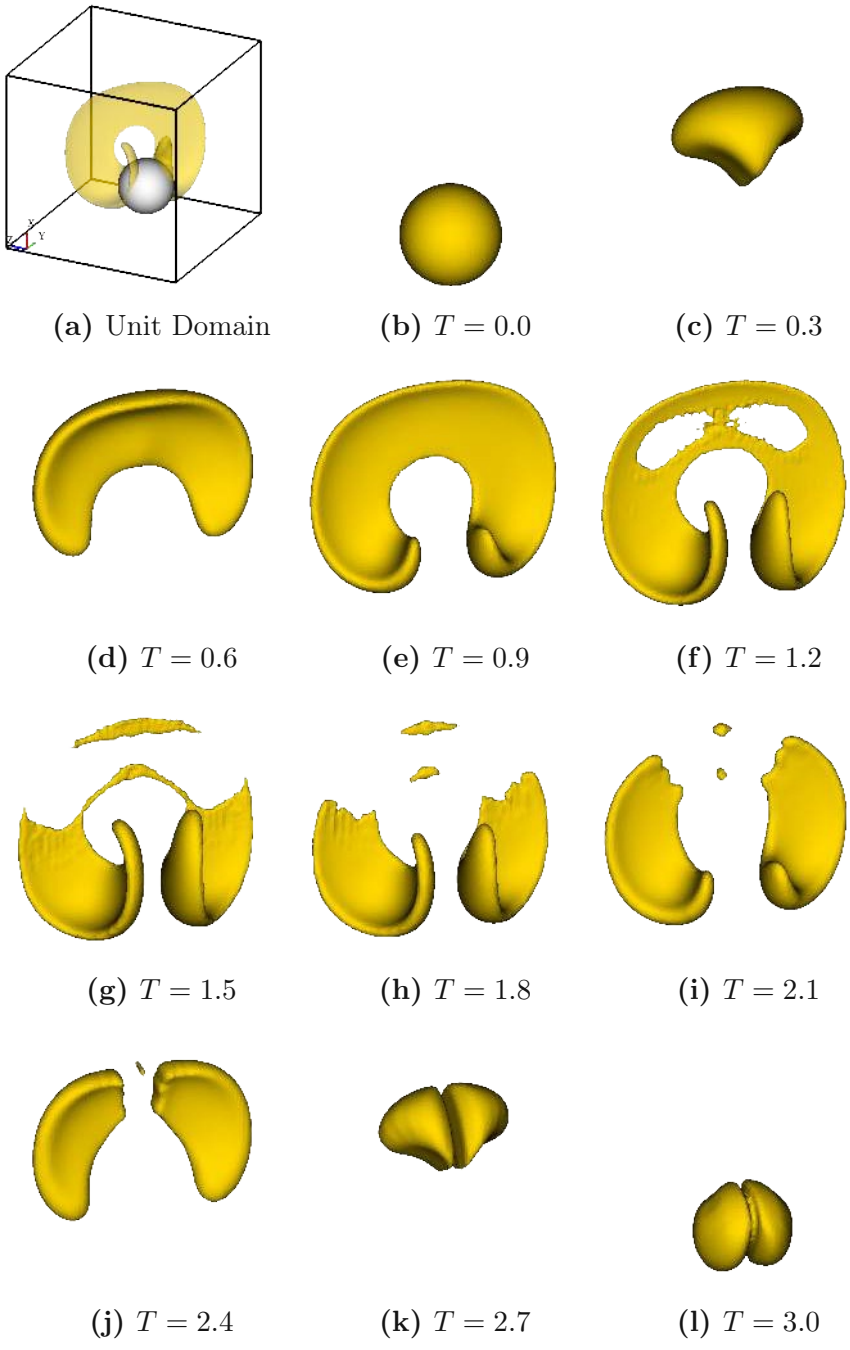
(a) Unit Domain **(b)** $T = 0.0$ **(c)** $T = 0.3$

(d) $T = 0.6$ **(e)** $T = 0.9$ **(f)** $T = 1.2$

(g) $T = 1.5$ **(h)** $T = 1.8$ **(i)** $T = 2.1$

(j) $T = 2.4$ **(k)** $T = 2.7$ **(l)** $T = 3.0$

Figure A.3: Results for the *Enright test* for resolution 256^3.



(a) Unit Domain **(b)** $T = 0.0$ **(c)** $T = 0.3$

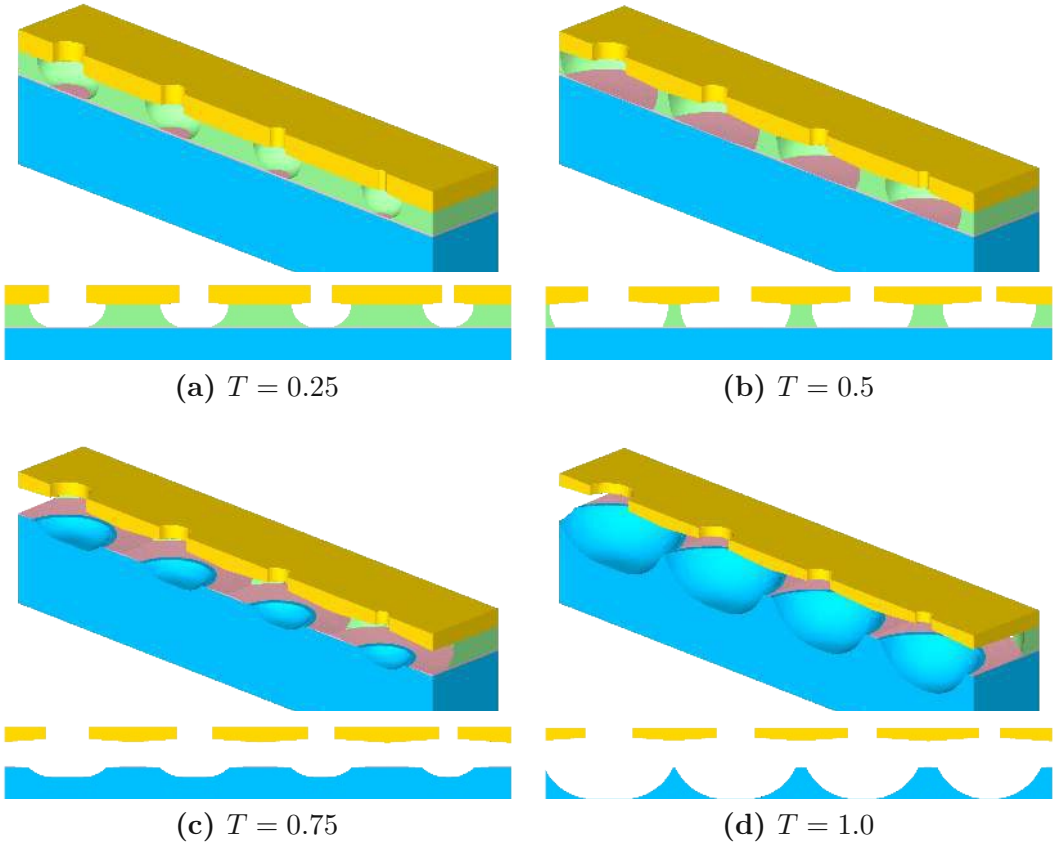
(d) $T = 0.6$ **(e)** $T = 0.9$ **(f)** $T = 1.2$

(g) $T = 1.5$ **(h)** $T = 1.8$ **(i)** $T = 2.1$

(j) $T = 2.4$ **(k)** $T = 2.7$ **(l)** $T = 3.0$

Figure A.4: Results for the *Enright test* for resolution 128^3.

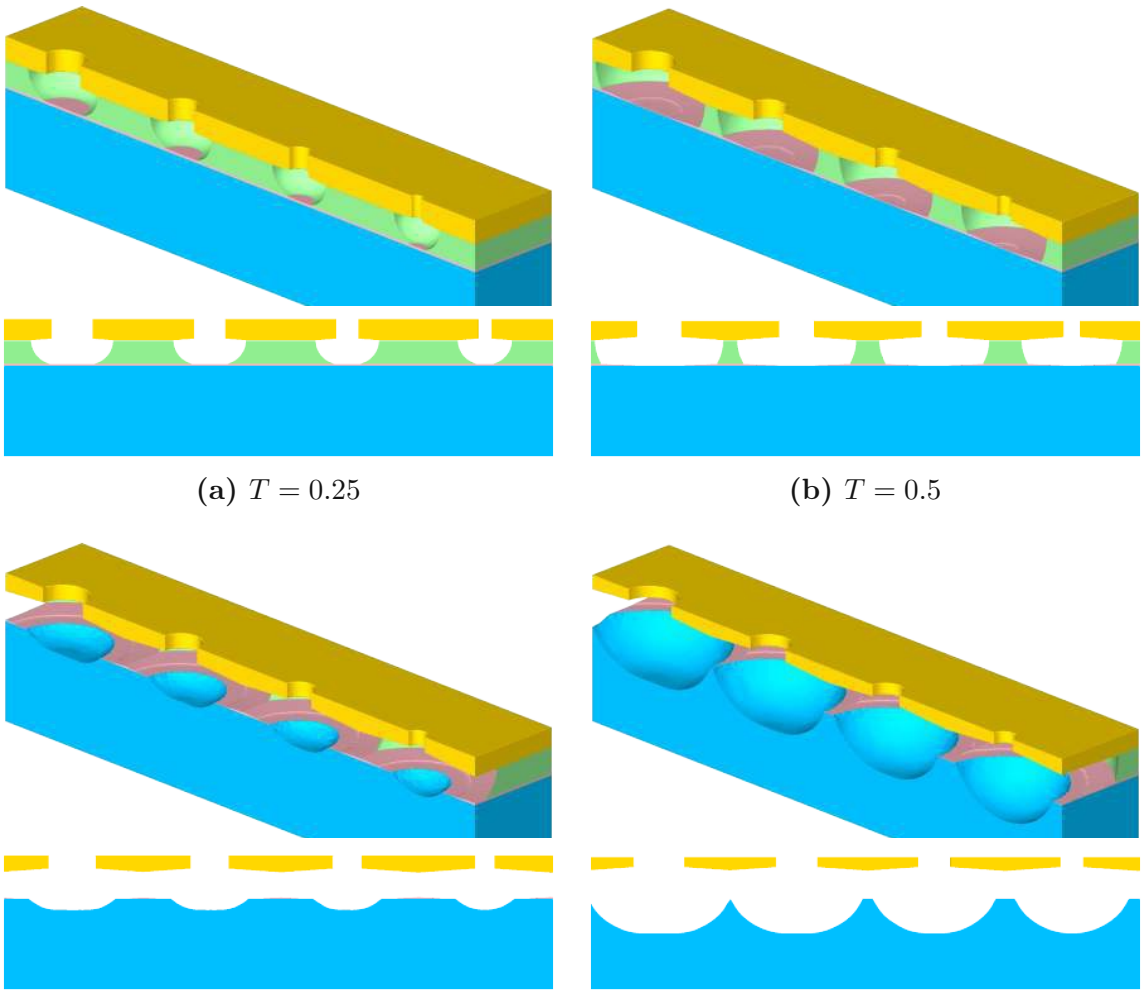
A.2 Material Dependent Isotropic Etching



(a) $T = 0.25$ **(b)** $T = 0.5$

(c) $T = 0.75$ **(d)** $T = 1.0$

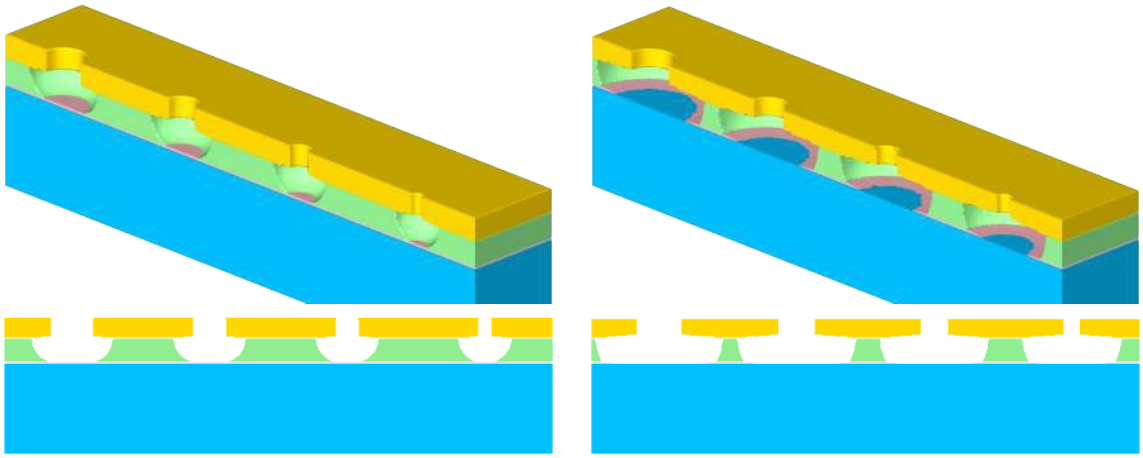
Figure A.5: Extracted material regions for resolution $1/256$ at $T = 0.25, 0.5, 0.75$, and 1.0. Regions thinner than one grid cell potentially vanish during the Boolean operations between the level-sets, which is visible in (c) and (d) near the edge of the crater in the blue material. The same results for resolutions $1/128$ and $1/64$ are provided in Figures A.6 and A.7.



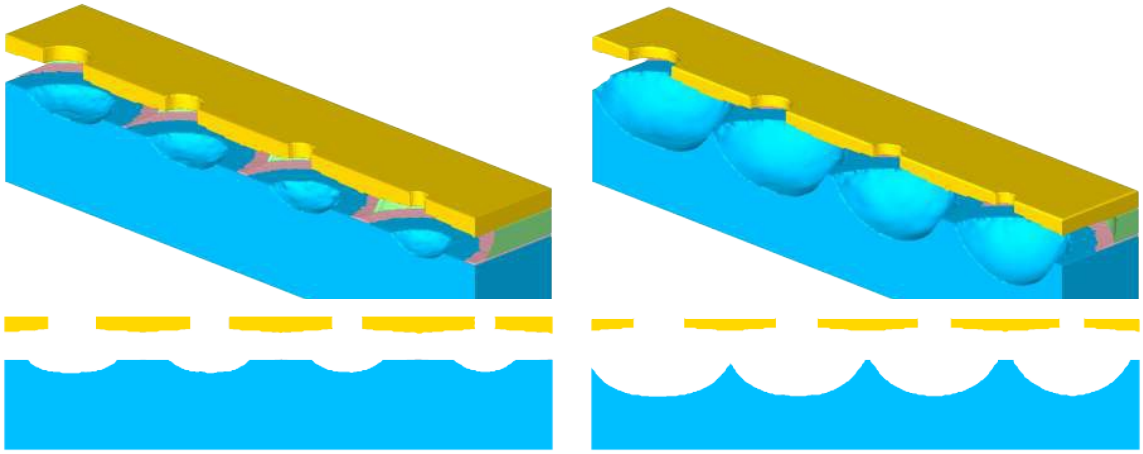
(a) $T = 0.25$ (b) $T = 0.5$

(c) $T = 0.75$ (d) $T = 1.0$

Figure A.6: Extracted material regions at $T = 0.25, 0.5, 0.75$, and 1.0 for resolution 1/128.



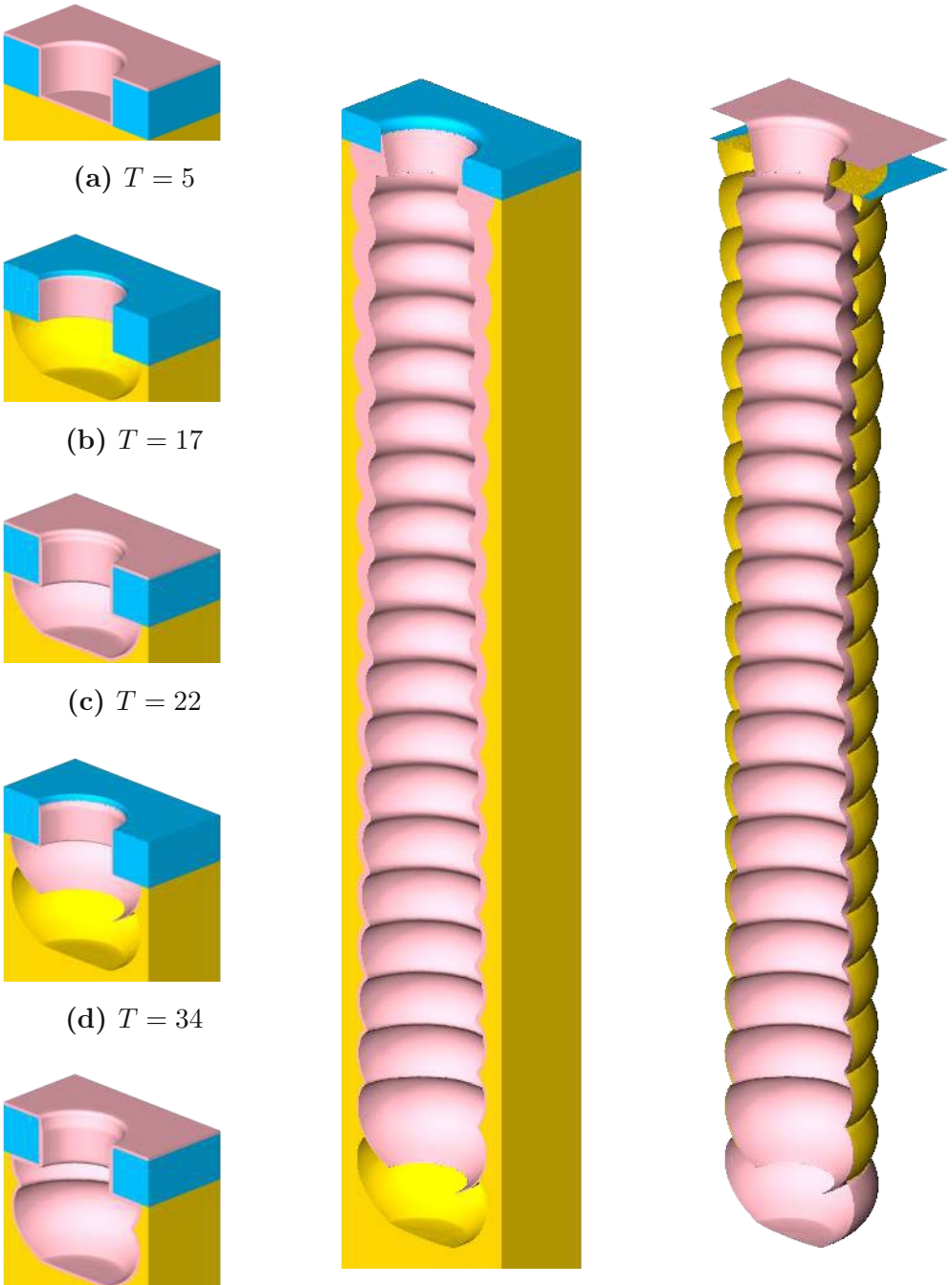
(a) $T = 0.25$ (b) $T = 0.5$



(c) $T = 0.75$ (d) $T = 1.0$

Figure A.7: Extracted material regions at $T = 0.25, 0.5, 0.75$, and 1.0 for resolution 1/64.

A.3 Simple Bosch Process



(a) $T = 5$

(b) $T = 17$

(c) $T = 22$

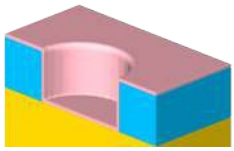
(d) $T = 34$

(e) $T = 39$

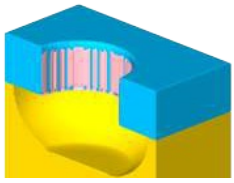
(f) $T = 340$, regions

(g) $T = 340$, layers

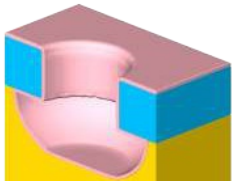
Figure A.8: Material regions of the simple Bosch process test case for resolution 1/256. (a)-(e): Material regions after each of the first 5 processing steps, i.e., alternating deposition and etching steps. (f): Final material regions after 20 cycles. (g): Final material layers after 20 cycles. The same results for lower resolutions 1/128, 1/64, and 1/32 are provided in Figures A.9-A.11.



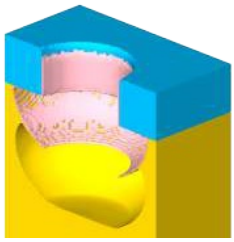
(a) $T = 5$



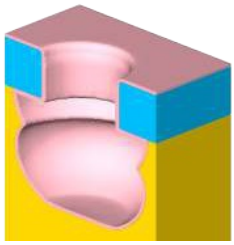
(b) $T = 17$



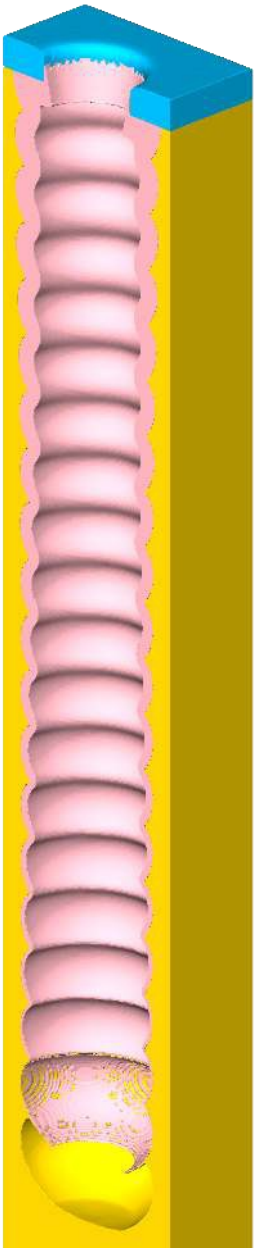
(c) $T = 22$



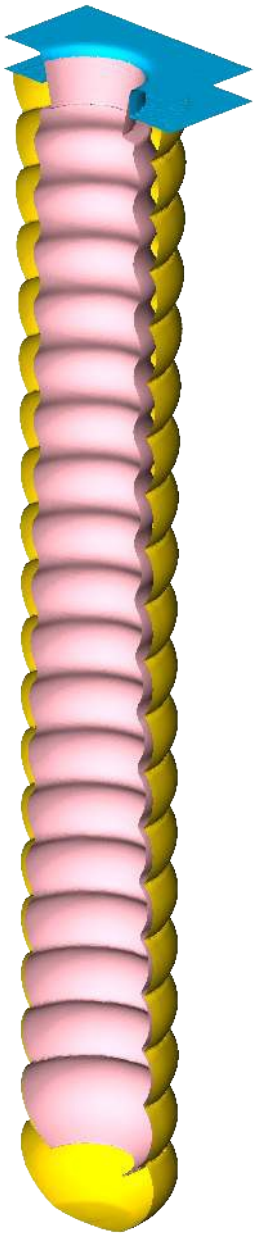
(d) $T = 34$



(e) $T = 39$

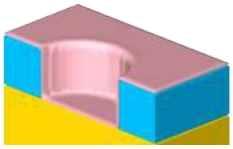


(f) $T = 340$, regions

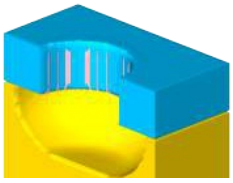


(g) $T = 340$, layers

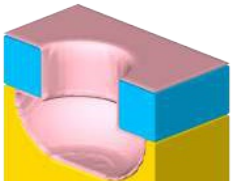
Figure A.9: Material regions of the simple Bosch process test case for resolution 1/128.



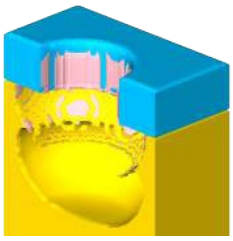
(a) $T = 5$



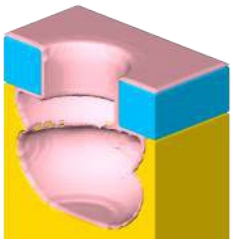
(b) $T = 17$



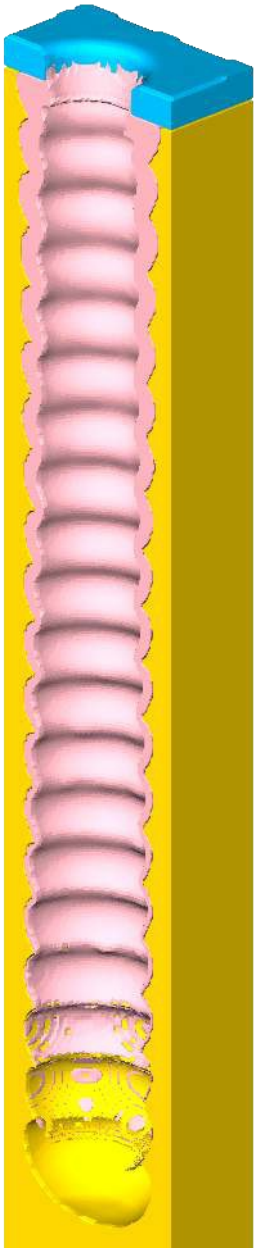
(c) $T = 22$



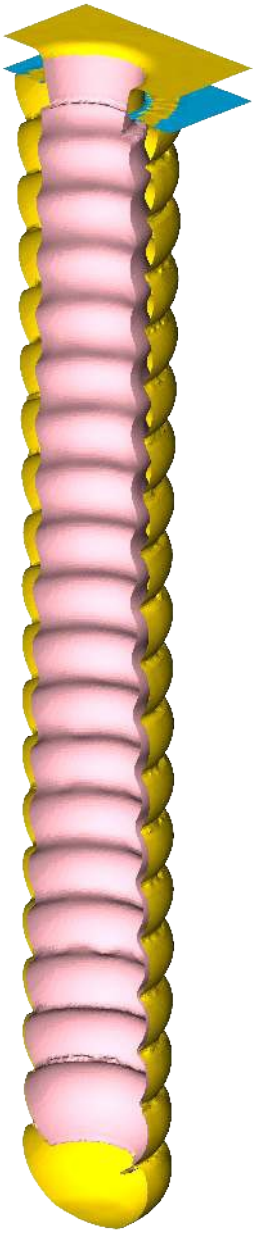
(d) $T = 34$



(e) $T = 39$

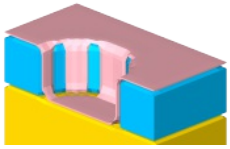


(f) $T = 340$, regions

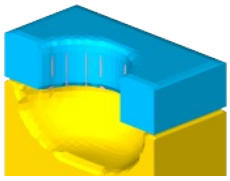


(g) $T = 340$, layers

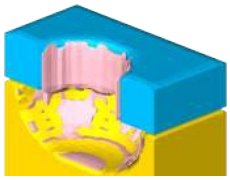
Figure A.10: Material regions of the simple Bosch process test case for resolution 1/64.



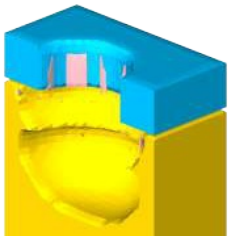
(a) $T = 5$



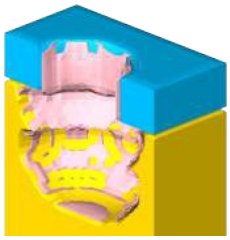
(b) $T = 17$



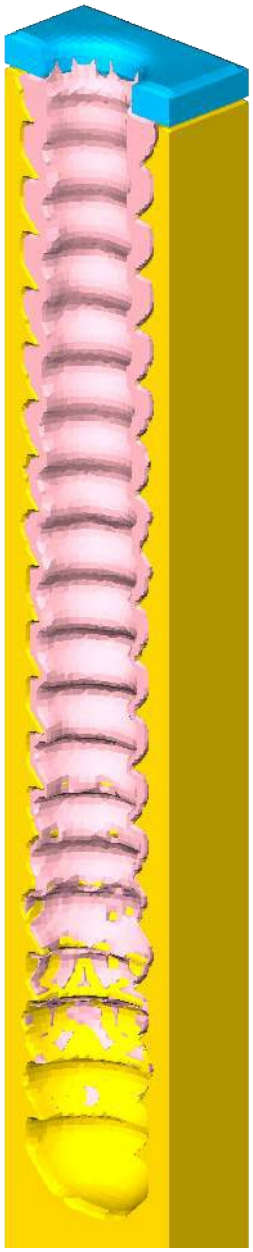
(c) $T = 22$



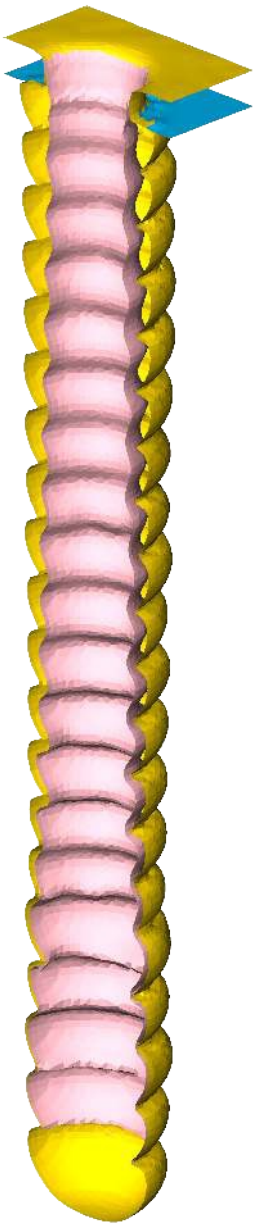
(d) $T = 34$



(e) $T = 39$



(f) $T = 340$, regions



(g) $T = 340$, layers

Figure A.11: Material regions of the simple Bosch process test case for resolution 1/32.

Appendix B

Supplementary Material Chapter 4

B.1 Analytical Solutions for Direct Flux from Power Cosine Sources

Direct flux originating from a source with flux distribution Γ_{src}

$$F_i = \int_{\Omega_{HS}} \Gamma_{src}(\boldsymbol{\Theta})(\boldsymbol{\Theta} \cdot \mathbf{n_i}) d\boldsymbol{\omega_\Theta}, \quad \text{with} \quad d\boldsymbol{\omega_\Theta} = sin\theta d\theta d\varphi \ .$$

Direct flux originating from a source with power cosine distribution

$$F_i = \int_{\Omega_{HS}} \left[\cos(\theta)^n \boldsymbol{\Theta}(\varphi,\theta) \cdot \mathbf{n_i}(\varphi,\theta)\right] sin\theta d\theta d\varphi \ .$$

Direct flux originating from a source with power cosine distribution on a horizontal surface, i.e., $\mathbf{n_i}(\varphi, 0)$

$$\begin{aligned} F_{horizontal} &= \int_0^{\pi/2} \int_0^{2\pi} \left[\cos(\theta)^n \underbrace{\boldsymbol{\Theta}(\varphi,\theta) \cdot \mathbf{n_i}(\varphi,0)}_{\cos(\theta)}\right] sin\theta d\theta d\varphi \\ &= \int_0^{\pi/2} \int_0^{2\pi} \left[\cos(\theta)^n cos(\theta)\right] sin\theta d\theta d\varphi \\ &= \int_0^{\pi/2} \int_0^{2\pi} \left[\cos(\theta)^{n+1}\right] sin\theta d\theta d\varphi = \frac{2}{n+2}\pi \ . \end{aligned}$$

E.g., for $n = 1$ (i.e., a diffuse source)

$$\begin{aligned} F_{horizontal} &= \int_0^{\pi/2} \int_0^{2\pi} \left[\cos(\theta)^2\right] sin\theta d\theta d\varphi \\ &= 2\pi \int_0^{\pi/2} \left[\cos(\theta)^2\right] sin(\theta) d\theta \\ &= 2\pi \left[-\frac{1}{3} cos(\pi/2)^3 + \frac{1}{3} cos(0)^3\right] = 2\pi \left[\frac{1}{3}\right] = \frac{2}{3}\pi \ . \end{aligned}$$

Direct flux originating from a source with power cosine distribution on a vertical surface, i.e., $\mathbf{n_i}(\varphi, \pi/2)$

$$
\begin{aligned}
F_{vertical} &= \int_0^{\pi/2} \int_0^{\pi} \left[\cos(\theta)^n \underbrace{\mathbf{\Theta}(\varphi,\theta) \cdot \mathbf{n_i}(\varphi, \pi/2)}_{\sin(\theta)\cos(\varphi-\pi/2)} \right] sin\theta d\theta d\varphi \\
&= \int_0^{\pi/2} \int_0^{\pi} \left[cos(\theta)^n (sin(\theta) cos(\varphi - \pi/2)) \right] sin(\theta) d\theta d\varphi \\
&= \int_0^{\pi/2} \int_0^{\pi} \left[cos(\theta)^n (sin(\theta) sin(\varphi)) \right] sin(\theta) d\theta d\varphi \\
&= \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta)(-cos(\pi)) - cos(\theta)^n sin(\theta)(-cos(0)) \right] sin(\theta) d\theta \\
&= \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta)(1) - cos(\theta)^n sin(\theta)(-1) \right] sin(\theta) d\theta \\
&= 2 \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta) \right] sin(\theta) d\theta \ .
\end{aligned}
$$

E.g., for $n = 1$ (i.e., a diffuse source)

$$
\begin{aligned}
F_{vertical} &= 2 \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta) \right] sin(\theta) d\theta \\
&= 2 \left[\frac{1}{3} \right] = \frac{2}{3} \ .
\end{aligned}
$$

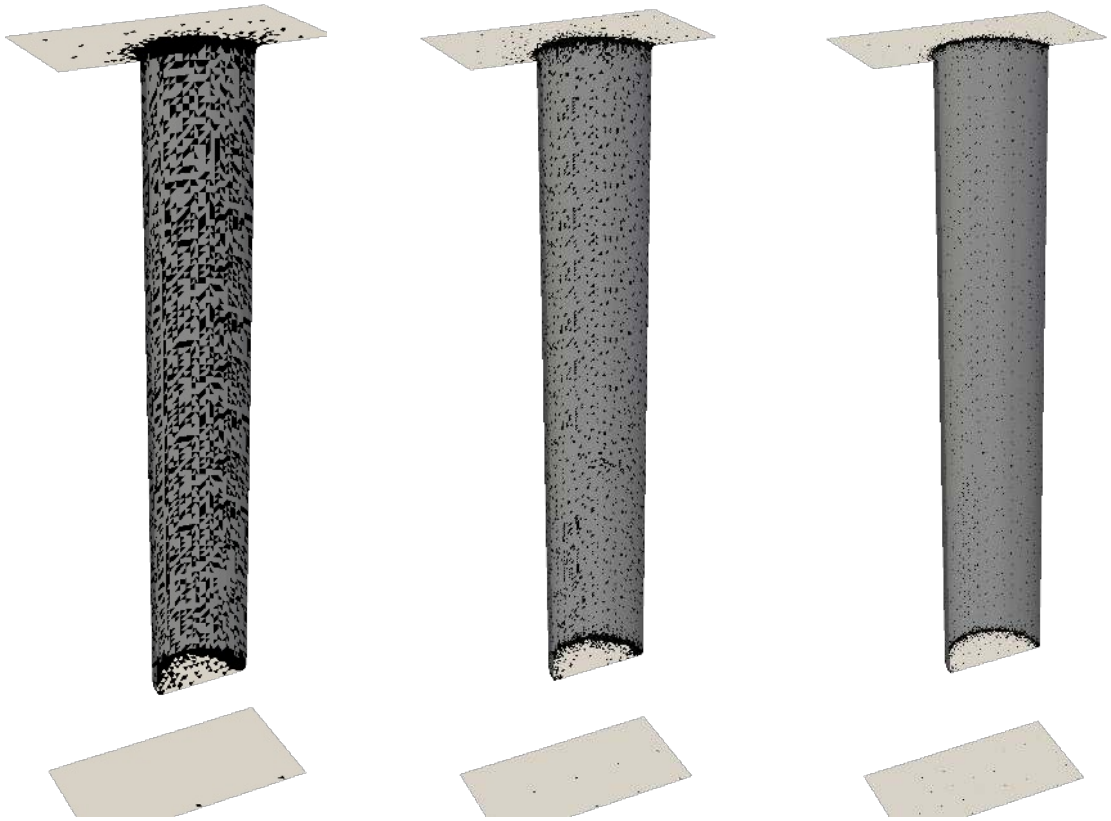
or other exponents

$$
\begin{aligned}
F_{vertical} &= 2 \int_0^{\pi/2} \left[cos(\theta)^n sin(\theta) \right] sin(\theta) d\theta \\
&= 2 \left[\frac{\pi}{16} \right] \quad (n = 2) \\
&= 2 \left[\frac{2}{15} \right] \quad (n = 3) \\
&= \dots \\
&= 2 \left[\frac{21\pi}{2048} \right] \quad (n = 10) \ .
\end{aligned}
$$

Appendix C

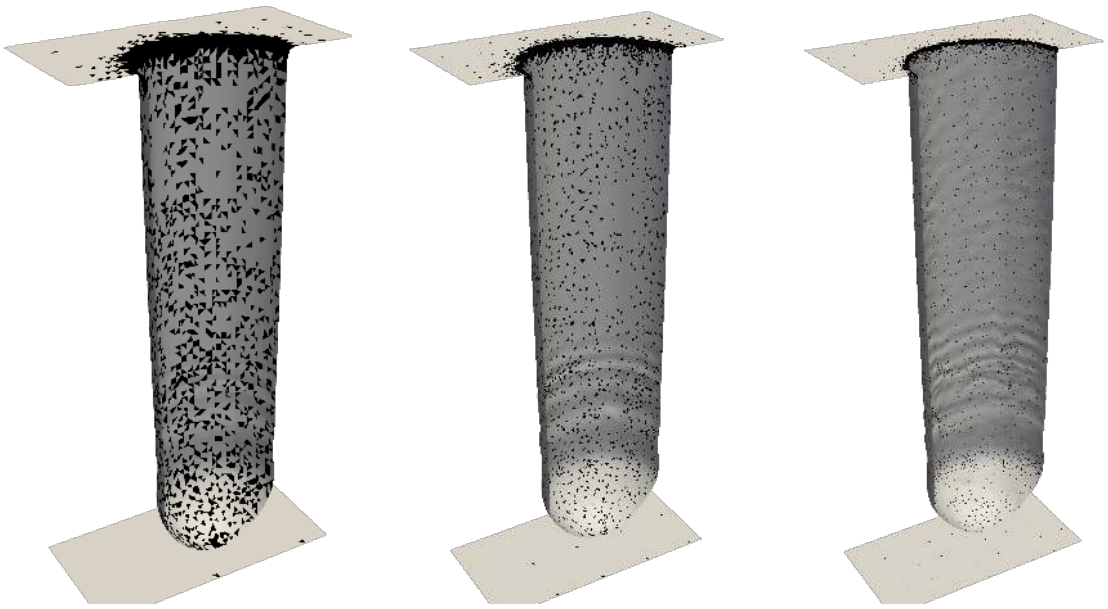
Supplementary Material Chapter 6

C.1 Additional Results



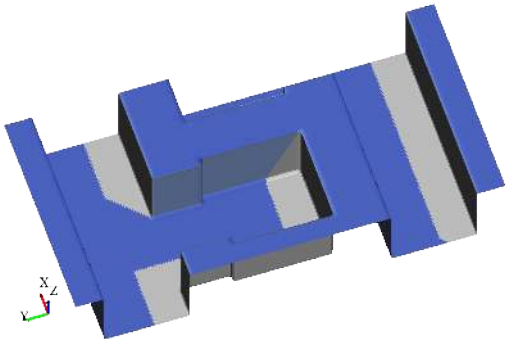
(a) 19456/4849 ≈ 4 **(b)** 71424/6221 ≈ 11 **(c)** 273664/8048 ≈ 34

Figure C.1: Sparse set of triangles (black) for $T = 0$, analog to Figure 6.4 in Chapter 6.

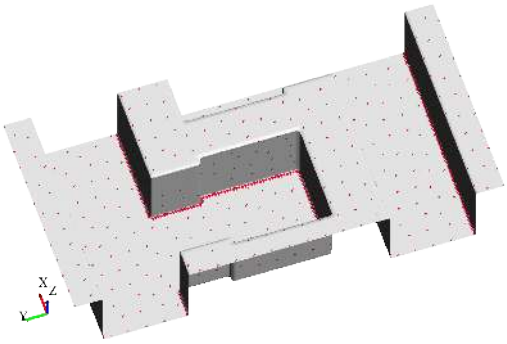


(a) $17840/3349 \approx 5$ **(b)** $64908/4717 \approx 14$ **(c)** $247892/7768 \approx 32$

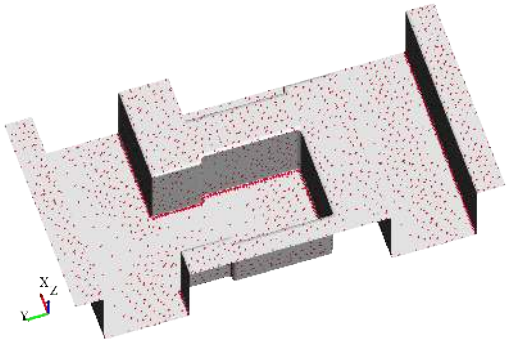
Figure C.2: Sparse set of triangles (black) for $T = 3$, analog to Figure 6.4 in Chapter 6.



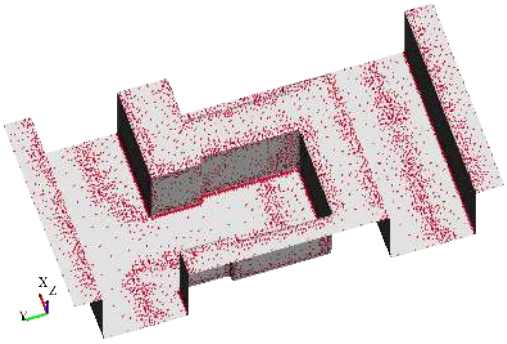
(a) direct flux, initial geometry



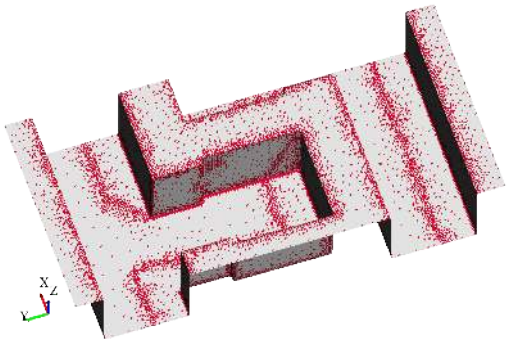
(b) $d_{max} = 16$



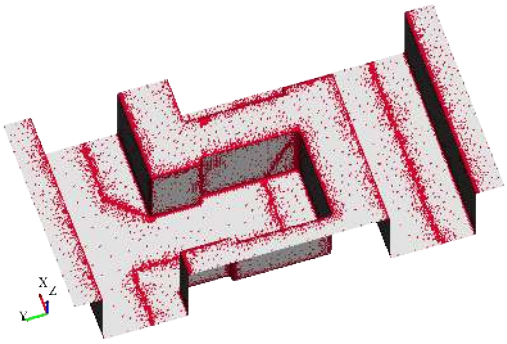
(c) $d_{max} = 8$



(d) $d_{max} = 4$



(e) $d_{max} = 2$



(f) $d_{max} = 1$

Figure C.3: Results for each iteration of the partitioning scheme ($d_{max_0} = 16$) for the geometry introduced in Figure 6.1. (a) Direct flux on the initial geometry. (b) Initial sparse set (including the material interfaces). (c)-(f) Results for the iterations $d_{max} = 8$ to $d_{max} = 1$.

C.2 Algorithm Subroutines

Algorithm 3: Recursive flagging and refinement of patches.

```
Function FlagNeighborhood(i, i_parent, i_prev, d_path, d_max):
    d_max_local = distTarget[i];
    if withdrawn[i] AND d_max >= d_path AND d_max_local > d_path AND
     distance[i] > d_path then
        touched[i] = true;
        parent[i] = i_parent;
        distance[i] = d_path;
        foreach i_ne in edgeNeighbors[i] do
            if i_ne != i_prev then
                FlagNeighborhood(i_ne, i_parent, i, d_path + 1, d_max)
    else
        SetNeighbors(i, i_parent);
Function FlagTriangles(indices, d_max):
    touched[] = false;
    d_path = 0;
    numNewPatches = 0;
    foreach i in indices do
        if !touched[i] and withdrawn[i] then
            ++numNewPatches;
            active[i] = touched[i] = reflagged[i] = true;
            parent[i] = i;
            distance[i] = d_path;
            foreach i_ne in edgeNeighbors[i] do
                FlagNeighborhood(i_ne, i, i, d_path + 1, d_max)
    return numNewPatches
Function RefinePatch(i_active, d_max):
    count = Withdraw(i_active, d_max/2)
    if count == 0 then
        return 0
    else
        UnSetAllNeighbors(i_active)
        numNewPatches = FlagTriangles(patches[i_active].patchIndices, d_max)
        RebuildNeighbors(i_active)
        UnWithdraw(i_active)
    return numNewPatches
```

Algorithm 4: Helper functions for sparse neighbor handling.

Function `SetNeighbors`(i, i_{active})**:**
- **if** *parent[i] != −1 and parent[i] !=* i_{active} **then**
  - sparseNeighbors[parent[i]].insert(i_{active});
  - sparseNeighbors[i_{active}].insert(parent[i]);

Function `UnSetAllNeighbors`(i_{active})**:**
- **foreach** i_{ns} *in sparseNeighbors[*i_{active}*].activeNeighbors* **do**
  - sparseNeighbors[i_{ns}].erase(i_{active});
  - sparseNeighbors[i_{active}].erase(i_{ns});

Function `RebuildNeighbors`(i_{active})**:**
- **foreach** i *in patches[*i_{active}*].patchIndices* **do**
  - **if** !*withdrawn*[i] **then**
    - **foreach** i_{ne} *in edgeNeighbors[i]* **do**
      - `SetNeighbors`(i_{ne}, i_{active});

Algorithm 5: Helper functions for withdrawal and building patch information.

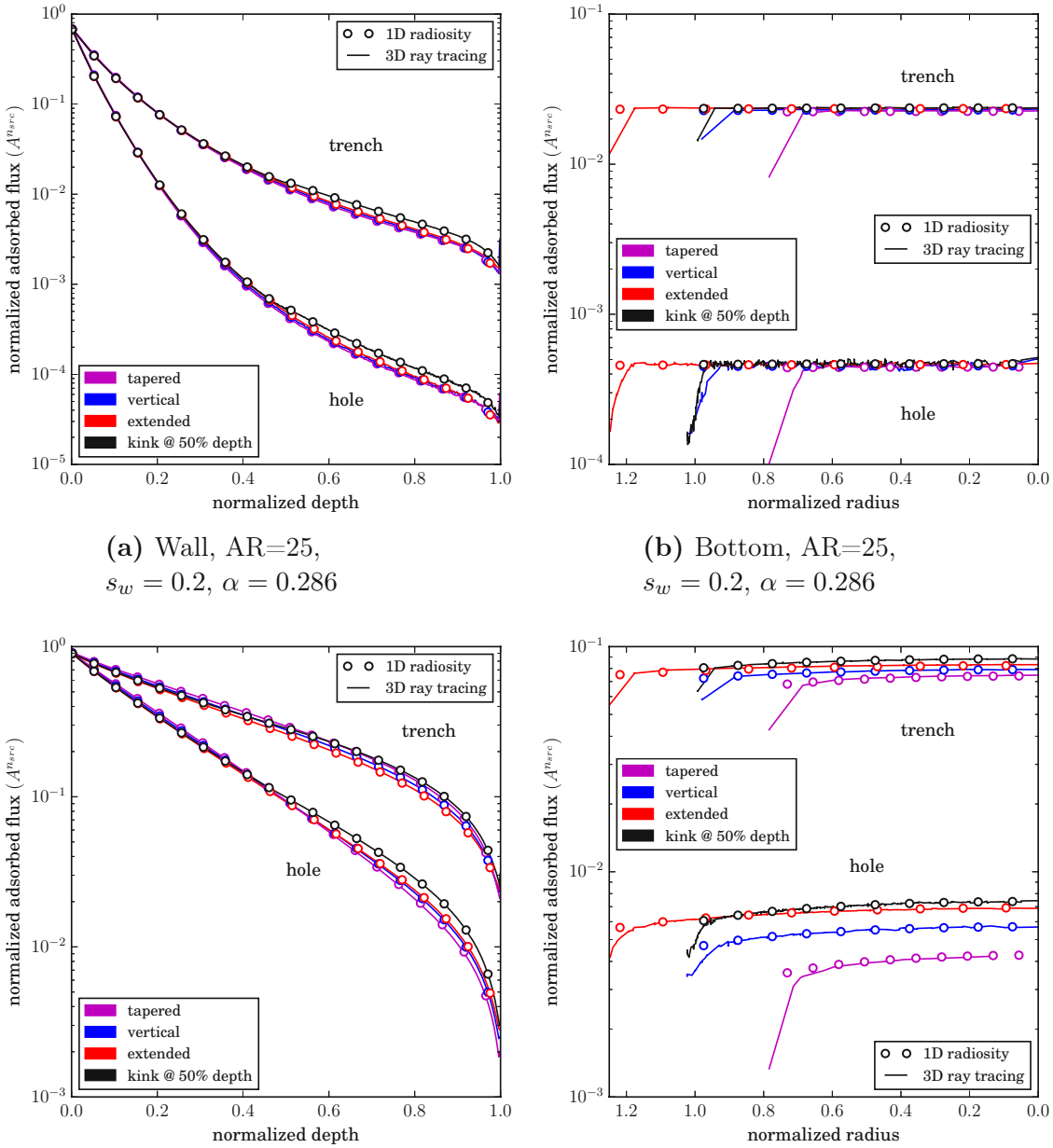
Function `Withdraw`(i_{active}, d)**:**
- count = 0;
- **foreach** i *in patches[*i_{active}*].patchIndices* **do**
  - **if** *distance*[i] > d **then**
    - withdrawn[i] = true;
    - distance[i] = d_{max_0};
    - parent[i] = -1;
    - ++count;
- **return** count

Function `UnWithdraw`(i_{active})**:**
- **foreach** i *in patches[*i_{active}*].patchIndices* **do**
  - withdrawn[i] = false;

Function `RebuildPatches`()**:**
- patches.clear();
- **for** $i = 0 \dots N_{tri} - 1$ **do**
  - **if** *parent[i] == -1* **then**
    - patches[parent[i]].insert(i);

Appendix D

Supplementary Material Chapter 7



(a) Wall, AR=25, $s_w = 0.2$, $\alpha = 0.286$

(b) Bottom, AR=25, $s_w = 0.2$, $\alpha = 0.286$

(c) Wall, AR=25, $s_w = 0.01$, $\alpha = 0.286$

(d) Bottom, AR=25, $s_w = 0.01$, $\alpha = 0.286$

Figure D.1: Normalized flux distributions along the wall and at the bottom of a hole and a trench of AR=25: (a) (b): Sticking probability $s_w = 0.2$. (c) (d) Sticking probability $s_w = 0.01$. The geometry of the structures is varied according to Figure 7.5b-7.5e. Lines represent the results of the reference ray tracing simulator *ViennaTS*. The deviations between ray tracing and radiosity towards the wall-bottom interface are due to the limited grid resolution of the ray tracing simulator. The flux distributions at the bottom span the interval [0.75,0] for the tapered structures and [1.25, 0] for the extended structures.

Bibliography

[1] Otmar Ertl and Siegfried Selberherr. "Three-Dimensional Level Set Based Bosch Process Simulations Using Ray Tracing for Flux Calculation". In: *Microelectronic Engineering* 87.1 (2010), pp. 20–29.

[2] W. Stanley. *Silicon Processing for the VLSI Era, Volume 4: Deep Submicron Process Technology*. Lattice Press, Sunset Beach, CA, 2002.

[3] J. Kriz, C. Angelkort, M. Czekalla, S. Huth, D. Meinhold, A. Pohl, S. Schulte, A. Thamm, and S. Wallace. "Overview of Dual Damascene Integration Schemes in Cu BEOL Integration". In: *Microelectronic Engineering* 85.10 (2008), pp. 2128–2132.

[4] Joyeeta Nag, Shishir Ray, Kriteshwar K. Kohli, Andrew H. Simon, Brian A. Cohen, Felipe Tijiwa-Birk, Christopher J. Parks, and Siddarth A. Krishnan. "Non-Contact, Sub-Surface Detection of Alloy Segregation in Back-End of Line Copper Dual-Damascene Structures". In: *IEEE Transactions on Semiconductor Manufacturing* 28.4 (2015), pp. 469–473.

[5] Otmar Ertl. "Numerical Methods for Topography Simulation". Doctoral Dissertation. TU Wien, 2010.

[6] Mohammad Reza Shaeri, Tien-Chien Jen, Chris Yingchun Yuan, and Masud Behnia. "Investigating atomic layer deposition characteristics in multi-outlet viscous flow reactors through reactor scale simulations". In: *International Journal of Heat and Mass Transfer* 89 (2015), pp. 468 –481.

[7] Frederic Gibou, Ronald Fedkiw, and Stanley Osher. "A Review of Level-Set Methods and Some Recent Applications". In: *Journal of Computational Physics* 353 (2017), pp. 82–109.

[8] Silvaco. *Victory Process - 3D Process Simulator*. URL: `http://www.silvaco.com/products/tcad/process_simulation/victory_process/victory_process.html`.

[9] J. Pagazani, F. Martyl, A. Babayan, A. Hoessinger, G. Lissorgues, and A. Nejim. "DRIE Process Modelling - A MEMS Case Study on a Real Design". In: *Proceedings of the 2013 Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS*. 2013, pp. 1–3.

[10] Synopsis. *Sentaurus Topography*. URL: `https://www.synopsys.com/silicon/tcad/process-simulation/sentaurus-topography.html`.

[11] Cheng-En Wu, Wayne Yang, Lan Luan, and Hua Song. "Photoresist 3D Profile Related Etch Process Simulation and its Application to Full Chip Etch Compact Modeling". In: *Proceedings of the SPIE Advanced Lithography Conference*. Vol. 9426. 2015, 94261Q:1–94261Q:8.

[12] Coventor. *SEMulator3D Advanced Modeling*. URL: `https://www.coventor.com/semiconductor-solutions/semulator3d/semulator3d-advanced-modeling/`.

[13] Aurelie Juncker, William Clark, Benjamin Vincent, Joern-Holger Franke, Sandip Halder, Frederic Lazzarino, and Gayle Murdoch. "Self-Aligned Block and Fully Self-Aligned Via for iN5 Metal 2 Self-Aligned Quadruple Patterning". In: *Proceedings of the SPIE Advanced Lithography Conference*. Vol. 10583. 2018, 105830W:1–105830W:11.

[14] Otmar Ertl, Lado Filipovic, Paul Manstetten, Xaver Klemenschits, and Josef Weinbub. *ViennaTS - The Vienna Topography Simulator*. URL: `https://github.com/viennats/viennats-dev`.

[15] Otmar Ertl and Siegfried Selberherr. "A Fast Level Set Framework for Large Three-Dimensional Topography Simulations". In: *Computer Physics Communications* 180.8 (2009), pp. 1242–1250.

[16] Alireza Sheikholeslami. "Topography Simulation of Deposition and Etching Processes". Doctoral Dissertation. TU Wien, 2004.

[17] Branislav Radjenović, Marija Radmilović-Radjenović, and Miodrag Mitrić. "Level Set Approach to Anisotropic Wet Etching of Silicon". In: *Sensors* 10.5 (2010), pp. 4950–4967.

[18] *ITK - Segmentation & Registration Toolkit*. URL: `https://itk.org/`.

[19] Jia-Cheng Yu, Zai-Fa Zhou, Jia-Le Su, Chang-Feng Xia, Xin-Wei Zhang, Zong-Ze Wu, and Qing-An Huang. "Three-Dimensional Simulation of DRIE Process Based on the Narrow Band Level Set and Monte Carlo Method". In: *Micromachines* 9.2 (2018), p. 74.

[20] Yiting Zhang. "Low Temperature Plasma Etching Control through Ion Energy Angular Distribution and 3-Dimensional Profile Simulation". PhD thesis. North Carolina State University, 2015.

[21] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, Burlington, 2016.

[22] James Arvo and David Kirk. "A Survey of Ray Tracing Acceleration Techniques". In: *An Introduction To Ray Tracing*. Academic Press, London, 1989, pp. 201–262.

[23] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. "Embree: A Kernel Framework for Efficient CPU Ray Tracing". In: *ACM Transactions on Graphics* 33.4 (2014), 143:1–143:8.

[24] Intel. *Embree*. URL: `https://embree.github.io/`.

[25] *nanort: NanoRT, single header only modern ray tracing kernel*. URL: `https://github.com/lighttransport/nanort`.

[26] Nvidia. *Optix/Optix Prime*. URL: https://developer.nvidia.com/download.

[27] Nvidia. *CUDA*. URL: https://developer.nvidia.com/cuda-zone.

[28] Matt Pharr. *pbrt-v3*. URL: https://github.com/mmp/pbrt-v3.

[29] *RadeonRays*. URL: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK.

[30] John C. Hart. "Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces". In: *The Visual Computer* 12.10 (1996), pp. 527–545.

[31] *OpenVDB - Sparse volume data structure and tools*. URL: https://github.com/dreamworksanimation/openvdb.

[32] Ken Museth. "Hierarchical Digital Differential Analyzer for Efficient Ray-Marching in OpenVDB". In: *Proceedings of the ACM SIGGRAPH*. 2014, p. 40.

[33] Matthias Müller. "Fast and Robust Tracking of Fluid Surfaces". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2009, pp. 237–245.

[34] Vittorio Cristini, Jerzy Bławzdziewicz, and Michael Loewenberg. "An Adaptive Mesh Algorithm for Evolving Surfaces: Simulations of Drop Breakup and Coalescence". In: *Journal of Computational Physics* 168.2 (2001), pp. 445–463.

[35] Andrei Zaharescu, Edmond Boyer, and Radu Horaud. "Topology-Adaptive Mesh Deformation for Surface Evolution, Morphing, and Multiview Reconstruction". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.4 (2011), pp. 823–837.

[36] Ernst Strasser and Siegfried Selberherr. "Algorithms and Models for Cellular Based Topography Simulation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14.9 (1995), pp. 1104–1114.

[37] 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". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.1 (2007), pp. 100–114.

[38] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, Cambridge, 1999.

[39] Jean-Christophe Nave, Rodolfo Ruben Rosales, and Benjamin Seibold. "A Gradient-Augmented Level Set Method with an Optimally Local, Coherent Advection Scheme". In: *Journal of Computational Physics* 229.10 (2010), pp. 3802–3827.

[40] Ebrahim M. Kolahdouz and David Salac. "A Semi-Implicit Gradient Augmented Level Set Method". In: *SIAM Journal on Scientific Computing* 35.1 (2013), A231–A254.

[41] Simone E. Hieber and Petros Koumoutsakos. "A Lagrangian Particle Level Set Method". In: *Journal of Computational Physics* 210.1 (2005), pp. 342–367.

[42] Rodolfo Bermejo and Juan Luis Prieto. "A Semi-Lagrangian Particle Level Set Finite Element Method for Interface Problems". In: *SIAM Journal on Scientific Computing* 35.4 (2013), A1815–A1846.

[43] Listy Stephen and Anoop Jose. "An Overview of Surface Tracking and Representation in Fluid Simulation". In: *International Journal of Advanced Computer Science and Applications* 6.11 (2015), pp. 281–286.

[44] Randall J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhäuser, Basel, 1990.

[45] Ravi Malladi, James A. Sethian, and Baba C. Vemuri. "Shape Modeling With Front Propagation: A level Set Approach". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.2 (1995), pp. 158–175.

[46] James A. Sethian. "A Fast Marching Level Set Method for Monotonically Advancing Fronts". In: *Proceedings of the National Academy of Sciences*. Vol. 93. 4. 1996, pp. 1591–1595.

[47] Li-Tien Cheng and Yen-Hsi Tsai. "Redistancing by Flow of Time Dependent Eikonal Equation". In: *Journal of Computational Physics* 227.8 (2008), pp. 4002–4017.

[48] M. W. Jones, J. A. Baerentzen, and M. Sramek. "3D Distance Fields: A Survey of Techniques and Applications". In: *IEEE Transactions on Visualization and Computer Graphics* 12.4 (2006), pp. 581–599.

[49] Hongkai Zhao. "A Fast Sweeping Method for Eikonal Equations". In: *Mathematics of Computation* 74.250 (2005), pp. 603–627.

[50] Stanley Osher and Chi-Wang Shu. "High-Order Essentially Nonoscillatory Schemes for Hamilton–Jacobi Equations". In: *SIAM Journal on Numerical Analysis* 28.4 (1991), pp. 907–922.

[51] Guang-Shan Jiang and Danping Peng. "Weighted ENO schemes for Hamilton–Jacobi equations". In: *SIAM Journal on Scientific Computing* 21.6 (2000), pp. 2126–2143.

[52] Ami Harten, Bjorn Engquist, Stanley Osher, and Sukumar R. Chakravarthy. "Uniformly High Order Accurate Essentially Non-Oscillatory Schemes". In: *Upwind and High-Resolution Schemes*. Springer, Berlin Heidelberg, 1987, pp. 218–290.

[53] Peter K. Sweby. "Godunov Methods". In: *Godunov Methods*. Springer, Boston, 2001, pp. 879–898.

[54] Olindo Zanotti and Gian Mario Manca. *A Very Short Introduction to Godunov Methods*. Lecture Notes for the COMPSTAR School on Computational Astrophysics. 2010.

[55] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Vol. 31. Cambridge University Press, Cambridge, 2002.

[56] Ami Harten. “High Resolution Schemes for Hyperbolic Conservation Laws”. In: *Journal of Computational Physics* 49.3 (1983), pp. 357–393.

[57] Chi-Wang Shu and Stanley Osher. “Efficient Implementation of Essentially Non-Oscillatory Shock-Capturing Schemes, II”. In: *Journal of Computational Physics* 83.1 (1989), pp. 32–78.

[58] Richard Courant, Kurt Friedrichs, and Hans Lewy. “Über die partiellen Differenzengleichungen der mathematischen Physik”. In: *Mathematische Annalen* 100.1 (1928), pp. 32–74.

[59] J. A. Sethian. “Fast Marching Methods and Level Set Methods for Propagating Interfaces”. In: *van Karman Institute Lecture Series*. Vol. 3. 1998, A1–A59.

[60] Ross T. Whitaker. “A Level-Set Approach to 3D Reconstruction from Range Data”. In: *International Journal of Computer Vision* 29.3 (1998), pp. 203–231.

[61] Chohong Min and Frédéric Gibou. “A Second Order Accurate Level Set Method on Non-Graded Adaptive Cartesian Grids”. In: *Journal of Computational Physics* 225.1 (2007), pp. 300–321.

[62] Ken Museth. “VDB: High-Resolution Sparse Volumes with Dynamic Topology”. In: *ACM Transactions on Graphics* 32.3 (2013), 27:1–27:22.

[63] Michael B. Nielsen and Ken Museth. “Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets”. In: *Journal of Scientific Computing* 26.3 (2006), pp. 261–299.

[64] Ben Houston, Michael B. Nielsen, Christopher Batty, Ola Nilsson, and Ken Museth. “Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation”. In: *ACM Transactions on Graphics* 25.1 (2006), pp. 151–175.

[65] Falko Löffler and Heidrun Schumann. “Generating Smooth High-Quality Isosurfaces for Interactive Modeling and Visualization of Complex Terrains”. In: *Proceedings of the International Workshop on Vision, Modeling and Visualization*. 2012.

[66] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. “Feature Sensitive Surface Extraction from Volume Data”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. 2001, pp. 57–66.

[67] Scott Schaefer and Joe Warren. “Dual Marching Cubes: Primal Contouring of Dual Grids”. In: *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*. 2004, pp. 70–76.

[68] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. “Dual Contouring of Hermite Data”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 339–346.

[69] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. “A Hybrid Particle Level Set Method for Improved Interface Capturing”. In: *Journal of Computational Physics* 183.1 (2002), pp. 83–116.

[70] Douglas Enright, Frank Losasso, and Ronald Fedkiw. "A Fast and Accurate Semi-Lagrangian Particle Level Set Method". In: *Computers & Structures* 83.6 (2005), pp. 479–490.

[71] Panagiotis Dimitrakis. *Charge-Trapping Non-Volatile Memories: Volume 1 – Basic and Advanced Devices*. Springer International Publishing, 2015.

[72] Peter Z. Kunszt, Alexander S. Szalay, and Aniruddha R. Thakar. "The Hierarchical Triangular Mesh". In: *Proceedings of the MPA/ESO/MPE Workshop Held at Garching*. Springer, 2001, pp. 631–637.

[73] Krzysztof M. Gorski, Eric Hivon, A. J. Banday, Benjamin D. Wandelt, Frode K. Hansen, Mstvos Reinecke, and Matthia Bartelmann. "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere". In: *The Astrophysical Journal* 622.2 (2005), p. 759.

[74] A. Vince. "Indexing a Discrete Global Grid". In: *Special Topics in Computing and ICT Research, Advances in Systems Modelling and ICT Applications* 2 (2006), pp. 3–17.

[75] Tamás Budavári, Alexander S. Szalay, and György Fekete. "Searchable Sky Coverage of Astronomical Observations: Footprints and Exposures". In: *Publications of the Astronomical Society of the Pacific* 122.897 (2010), p. 1375.

[76] Max Tegmark. "An Icosahedron-Based Method for Pixelizing the Celestial Sphere". In: *The Astrophysical Journal* 470 (1996), pp. L81–L84.

[77] Chihiro Kodama, Masaaki Terai, Akira T. Noda, Yohei Yamada, Masaki Satoh, Tatsuya Seiki, Shin-ichi Iga, Hisashi Yashiro, Hirofumi Tomita, and Kazuo Minami. "Scalable Rank-Mapping Algorithm for an Icosahedral Grid System on the Massive Parallel Computer with a 3-D Torus Network". In: *Parallel Computing* 40.8 (2014), pp. 362–373.

[78] Alexander S. Szalay, Jim Gray, George Fekete, Peter Z. Kunszt, Peter Kukol, and Ani Thakar. *Indexing the Sphere with the Hierarchical Triangular Mesh*. Technical Report MSR-TR-2005-123, Microsoft Research. 2005.

[79] Paul Manstetten, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. "Accelerated Direct Flux Calculations Using an Adaptively Refined Icosahedron". In: *Proceedings of the 22nd International Conference on Simulation of Semiconductor Processes and Devices*. 2017, pp. 73–76.

[80] Paul Manstetten, Josef Weinbub, Andreas Hössinger, and Siegfried Selberherr. "Using Temporary Explicit Meshes for Direct Flux Calculation on Implicit Surfaces". In: *Procedia Computer Science* 108 (2017), pp. 245–254.

[81] Kendall Atkinson. "Numerical Integration on the Sphere". In: *The Journal of the Australian Mathematical Society. Series B. Applied Mathematics* 23.03 (1982), pp. 332–347.

[82] John C. Hart. "Ray Tracing Implicit Surfaces". In: *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces* (1993), pp. 1–16.

[83] Paul Manstetten, Lukas Gnam, Andreas Hössinger, Siegfried Selberherr, and Josef Weinbub. “Sparse Surface Speed Evaluation on a Dynamic Three-Dimensional Surface Using an Iterative Partitioning Scheme”. In: *Lecture Notes in Computer Science*. accepted, in print.

[84] I.N. Bronshtein, K.A. Semendyayev, G. Musiol, and H. Mühlig. *Handbook of Mathematics*. Fifth Ed. Springer, Berlin Heidelberg, 2007.

[85] George Kokkoris, Andreas G. Boudouvis, and Evangelos Gogolides. “Integrated Framework for the Flux Calculation of Neutral Species Inside Trenches and Holes During Plasma Etching”. In: *Journal of Vacuum Science & Technology A* 24.6 (2006), pp. 2008–2020.

[86] Evert J. Nyström. “Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben”. In: *Acta Mathematica* 54.1 (1930), pp. 185–204.

[87] Paul Manstetten, Lado Filipovic, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. “Framework to Model Neutral Particle Flux in Convex High Aspect Ratio Structures using One-Dimensional Radiosity”. In: *Solid-State Electronics* 128.2 (2017), pp. 141–147.

[88] Michael F Modest. *Radiative Heat Transfer*. Academic Press, 2013.

[89] John R Howell, M Pinar Menguc, and Robert Siegel. *Thermal Radiation Heat Transfer*. CRC press, 2010.

[90] Jane P. Chang Francis F. Chen. *Lecture Notes on Principles of Plasma Processing*. Springer, 2003.

Own Publications

Journal Articles

[1] Paul Manstetten, Josef Weinbub, Andreas Hössinger, and Siegfried Selberherr. "Using Temporary Explicit Meshes for Direct Flux Calculation on Implicit Surfaces". In: *Procedia Computer Science* 108 (2017), pp. 245–254.

[2] Paul Manstetten, Lado Filipovic, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. "Framework to Model Neutral Particle Flux in Convex High Aspect Ratio Structures using One-Dimensional Radiosity". In: *Solid-State Electronics* 128.2 (2017). invited, pp. 141–147.

Book Contributions

[3] Paul Manstetten, Lukas Gnam, Andreas Hössinger, Siegfried Selberherr, and Josef Weinbub. "Sparse Surface Speed Evaluation on a Dynamic Three-Dimensional Surface Using an Iterative Partitioning Scheme". In: *Lecture Notes in Computer Science*. accepted, in print.

[4] Paul Manstetten, Lado Filipovic, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. "Using One-Dimensional Radiosity to Model Neutral Particle Flux in High Aspect Ratio Holes". In: *Proceedings of the 2016 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon (EUROSOI-ULIS)*. IEEE Xplore, 2016, pp. 120–123.

Conference Contributions

[5] Paul Manstetten, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. "Accelerated Direct Flux Calculations Using an Adaptively Refined Icosahedron". In: *Proceedings of the 22nd International Conference on Simulation of Semiconductor Processes and Devices*. Talk given in Kamakura, Japan, 2017, pp. 73–76.

[6] Paul Manstetten, Lado Filipovic, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. "Using One-Dimensional Radiosity to Model Neutral Flux in Convex High Aspect Ratio Structures". In: *Proceedings of the 21st International Conference on Simulation of Semiconductor Processes and Devices*. Poster presented in Nürnberg, Germany, 2016, pp. 265–268.

[7] Paul Manstetten, Lado Filipovic, Andreas Hössinger, Josef Weinbub, and Siegfried Selberherr. “Modeling Neutral Particle Flux in High Aspect Ratio Holes using a One-Dimensional Radiosity Approach”. In: *Book of Abstracts of the 2016 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon*. Talk given in Vienna, 2016, pp. 68–69.

[8] Paul Manstetten, Vito Simonka, Georgios Diamantopoulos, Lukas Gnam, Alexander Makarov, Andreas Hössinger, and Josef Weinbub. “Computational and Numerical Challenges in Semiconductor Process Simulation”. In: *CSE17 Abstracts*. Talk given by Josef Weinbub at the SIAM Conference on Computational Science and Engineering, Atlanta, GA, USA; 2017-02-27 – 2017-03-03. 2017, p. 46.

Curriculum Vitae

Personal Information

Name	Paul Ludwig Manstetten
Address	Pezzlgasse 14/8A
	1170 Wien
Phone	+43 660 740 1624
Email	paul@remans.de
Date of Birth	September 17, 1984, Berlin
Nationality	German

Education

10/2015	-	present	Doctoral Program, *Electrical Engineering*, *Institute for Microelectronics*, *Technische Universität (TU) Wien*
10/2010	-	09/2012	Graduate Studies, *Computational Engineering*, *Friedrich-Alexander Universität (FAU) Erlangen-Nürnberg*, *Faculty of Engineering*
10/2005	-	03/2010	Graduate Studies, *Mechatronics*, *Fachhochschule (FH) Regensburg*, *Faculty of Electrical Engineering and Information Technology*

Research Positions

09/2015	-	present	Project Assistant, *Christian Doppler Laboratory for High Performance TCAD*, *Institute for Microelectronics*, *TU Wien*
05/2012	-	09/2012	Research Assistant, *Department of Computer Science 10*, *FAU Erlangen-Nürnberg*
02/2012	-	05/2012	Research Assistant, *Chair of Sensor Technology*, *FAU Erlangen-Nürnberg*
10/2009	-	03/2010	Research Assistant, *LaS3*, *FH Regensburg*

Industry Positions

11/2012	-	09/2015	Application Engineer for Optical Simulations, *OSRAM Opto Semiconductors GmbH*, Regensburg
03/2008	-	07/2008	Intern (Advanced Development Group), *Continental Automotive GmbH*, Regensburg
10/2006	-	02/2007	Intern (Technical Lab), *BMW AG*, Regensburg
06/2005	-	08/2005	Intern, *Infineon AG*, Regensburg