TU TECHNISCHE UNIVERSITÄT WIEN WIEN

Diploma Thesis

# GPU-Accelerated Ray-Tracing for Particle Simulations in ViennaPS

submitted in satisfaction of the requirements for the degree

Master of Science

of the TU Wien, Faculty of Electrical Engineering and Information Technology

---

Diplomarbeit

# GPU-beschleunigtes Ray-Tracing für Partikelsimulationen in ViennaPS

ausgeführt zum Zwecke der Erlangung des akademischen Grads

Master of Science

eingereicht an der TU Wien, Fakultät für Elektrotechnik und Informationstechnik

**Xaver Riedel**, BSc

Matr.Nr.: 11922639

Betreuung: Prof. Dipl.-Ing. Dr.techn. **Lado Filipovic**
Dipl.-Ing. **Tobias Reiter**, BSc
Institut für Mikroelektronik
Forschungsbereich Mikroelektronik
Technische Universität Wien
Karlsplatz 13, 1040 Wien, Österreich

Wien, im Januar 2026

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Kurzfassung

Die Ray-Tracing-Komponente von ViennaPS, die zuvor eine CPU-Implementierung mit Unterstüzung für scheibenbasierte Oberflächen in 2D und 3D, sowie eine GPU-Implementierung die 3D-dreiecksbasierte Oberflächen unterstützte, wurde um drei zusätzliche GPU-beschleunigte Raytracing-Engines erweitert. Die neuen Engines unterstützen Scheiben in 2D und 3D, ähnlich der CPU-Version, 2D-Linien als Gegenstück zu den 3D-Dreiecks-Elementen sowie implizite Flächen in 2D und 3D, definiert durch ein Level-Set. Das Ziel ist es, Raytracing-Engines zu entwickeln und zu benchmarken, die Partikelsimulationen im Vergleich zur CPU erheblich beschleunigen, während Unterschiede und Kompromisse verschiedener Oberflächendarstellungen untersucht werden. Die GPU-Implementierungen wurden unter Verwendung des NVIDIA® OptiX™ Frameworks entwickelt, das Hardware-Beschleunigung über spezialisierte RT-Kerne moderner GPUs unterstützt. Diese RT-Kerne sind für das durchlaufen der BVH Struktur und Ray-Dreieck-Treffertests optimiert. Die Raytracing-Engines wurden verwendet, um Flussberechnungen während der Simulation mehrerer Halbleiter-Fertigungsprozesse zu beschleunigen. Speziell getestete Prozesse umfassen chemische Beschichtung, Plasmaätzen und den Bosch-Prozess. Die Methoden wurden hinsichtlich Gesamtlaufzeit und Genauigkeit mit der bestehenden CPU-Scheiben-Implementierung verglichen, wobei sowohl qualitative Analysen der finalen Oberflächen durch visuelle Vergleiche als auch quantitative Metriken wie Flächeninhalt und Chamfer-Distanz zur Diskussion der Genauigkeit herangezogen wurden. Die GPU-Scheiben-Engine erreichte 9- bis 34-fache Beschleunigungen, abhängig vom Beispiel, im Vergleich zur CPU-Version, während praktisch identische Ergebnisse produziert wurden. Die 2D-Linien-Engine erreichte ähnliche Leistungen, während die Dreiecks-Engine etwa 1,5-mal mehr Rays pro Sekunde verfolgen konnte, da die RT-Kerne speziell für Ray-Dreieck-Treffer ausgelegt sind. Da Dreiecksnetze in der Regel etwa doppelt so viele Oberflächenelemente wie ein Scheibennetz benötigen, um dieselbe Geometrie darzustellen, und die Anzahl der Rays in dieser Implementierung mit der Anzahl an Elementen skaliert, waren die Gesamtlaufzeiten vergleichbar. Es wurde auch versucht, eine explizite Oberflächendarstellung zu vermeiden und stattdessen direkt die Level-Set Repräsentation der Oberfläche zu verwenden, die von ViennaLS bereitgestellt wird. Die Genauigkeit war jedoch im Vergleich zu den netzbasierten Ansätzen nicht zufriedenstellend. Wahrscheinliche Ursache hierfür ist die einfache und fehleranfällige Implementierung zur Berechnung der Oberfläche der einzelnen Elemente, für die möglicherweise bessere Ansätze vorgeschalgen werden. Die Leistung der Level-Set-Engine war ebenfalls nicht auf dem Niveau der anderen Engines, da kein Aufwand unternommen wurde, die fehlerhafte Implementierung zu optimieren. Theoretisch sollte die Level-Set-Methode in der Lage sein, ähnliche Leistungen zu erzielen und dabei den Zwischenschritt der Netzgenerierung zu vermeiden, jedoch ist weitere Arbeit erforderlich, um dieses Ziel zu erreichen. Die GPU-Engines wurden alle mit NVIDIA® Nsight Compute™ analysiert, um Bereiche für zukünftige Optimierungen zu identifizieren, wobei potenzielle Verbesserungen in Speicherzugriffsmustern sowie Speicher- und Warp-Kohärenz diskutiert wurden. Schließlich wurde der Energieverbrauch von CPU und GPU während einer Simulation gemessen, um die Energieeffizienz der verschiedenen Implementierungen zu bewerten. Dabei zeigte sich, dass die GPU-Engines trotz erhöhter Gesamtleistung aufgrund der deutlich kürzeren Laufzeiten erheblich energieeffizienter sind.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Abstract

The ray tracing component of ViennaPS, which previously included a CPU implementation supporting disk-based surface meshes in 2D and 3D, and a GPU implementation supporting 3D triangle-based meshes, was extended by three additional GPU-accelerated ray tracing engines. The new engines add support for disk meshes in both 2D and 3D, matching the capabilities of the CPU version, 2D line meshes as counterparts to the 3D only triangle primitives and the use of 2D and 3D level-set voxel meshes. The goal is to create and benchmark ray tracing engines that significantly speed up particle simulations compared to the CPU, while exploring the effects and trade-offs of different surface representations. The GPU implementations were developed using the NVIDIA® OptiX™ framework, which leverages hardware acceleration through specialized RT cores built for optimized BVH traversal and ray-triangle intersection tests on modern NVIDIA GPUs. The ray tracing engines were used to accelerate flux calculations during the simulation of several semiconductor fabrication processes. Specifically, benchmarked processes include chemical deposition, plasma etching, and the Bosch process. The methods were compared in terms of total runtime and accuracy against the existing CPU disk implementation, while both qualitative analysis of final surfaces through visual comparison and quantitative metrics like surface area and Chamfer distance were used to discuss accuracy.
The GPU disk engine achieved speedups of 9x-34x, depending on the example, compared to the CPU version while producing practically identical results. The 2D line engine reached similar performance, while the triangle engine was able to trace around 1.5x more rays per second since the RT cores are specifically designed for ray-triangle intersection. Since triangle meshes typically require around twice as many surface elements as disk meshes to represent the same geometry and because the number of rays in this implementation scales with the number of elements, the overall runtimes were comparable. There was also an attempt to avoid explicit meshing and directly use the level-set representation of the surface provided by ViennaLS. However, the accuracy was not satisfactory compared to the mesh based approaches. A likely cause for this is the simple and error-prone implementation for surface element area calculation, for which possibly better approaches are mentioned. The performance of the level-set engine was also not competitive with the other engines, as no effort was made to optimize the erroneous implementation. In theory, the level-set method should be able to achieve similar performance while avoiding the intermediate meshing step; however, further work is required to reach this goal. The GPU engines were all profiled using NVIDIA® Nsight Compute™ to identify bottlenecks and areas for future optimization, where potential improvements in memory access patterns as well as memory and warp coherence were discussed. Finally, the power consumption of the CPU and GPU during a simulation was measured to evaluate energy efficiency of the different implementations, showing that the GPU engines are significantly more energy efficient despite their higher power draw due to their much shorter runtimes.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# Contents


**1 Introduction** **8**
1.1 Motivation and Goal . . . 8
1.2 Procedure and expected Outcome . . . 8

**2 TCAD Simulations** **10**
2.1 Historical Background of TCAD . . . 10
2.2 State of the Art Simulators . . . 10
2.2.1 Atomic Scale Simulations . . . 11
2.2.2 Feature Scale Simulations . . . 11
2.2.3 Device Scale Simulations . . . 12
2.3 Trends and Future Directions in TCAD . . . 12
2.3.1 Hardware Accelerators . . . 12
2.3.2 Machine Learning in TCAD . . . 13
2.4 Physical Process Simulation using ViennaPS . . . 14

**3 Fabrication Processes** **16**
3.1 Material Deposition . . . 16
3.2 Etching . . . 17
3.2.1 $SF_6/O_2$ Plasma Etching . . . 18
3.2.2 Bosch Process . . . 19
3.2.3 $HBr/O_2$ Plasma Etching . . . 20

**4 Process Simulation** **22**
4.1 Level-Set Surface Representation . . . 22
4.2 Particle-Particle Interaction . . . 24
4.3 Source Distribution . . . 24
4.4 Particle-Surface Interactions . . . 25
4.4.1 Simple Deposition/Etching Models . . . 25
4.4.2 Plasma Etching Models . . . 27

**5 Implementation** **29**
5.1 GPU-accelerated Ray Tracing Frameworks . . . 29
5.2 NVIDIA® OptiX™ . . . 30
5.2.1 Bounding Volume Hierarchy Acceleration Structure . . . 30
5.2.2 Shader Binding Table and Pipeline Programs . . . 31
5.3 Meshing and Axis-Aligned Bounding Boxes . . . 32
5.3.1 Disk Mesh . . . 33
5.3.2 Line/Triangle Mesh . . . 34
5.3.3 Level-Set Mesh . . . 35
5.4 Ray-Generation (RG) . . . 36
5.5 Ray-Element-Intersection (IS) . . . 37
5.5.1 Ray-Disk Intersection . . . 37
5.5.2 Ray-Line Intersection . . . 39



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.5.3 Ray-Level-Set Intersection . . . . . 40
5.6 Ray-Surface-Interaction (CH) . . . . . 43
5.6.1 Boundary Hits . . . . . 43
5.6.2 Backside Hits . . . . . 43
5.6.3 Direct Callables (DC) . . . . . 44
5.6.4 Warp Divergence and Shader Execution Reordering . . . . . 45
5.7 Post-Processing and Advection . . . . . 45
5.8 Limitations and Potential Improvements . . . . . 47

**6 Results** **48**
6.1 Comparison of CPU and GPU Disk Method . . . . . 48
6.2 Single Particle Deposition . . . . . 50
6.2.1 Scalability with Number of Surface Elements and Rays . . . . . 53
6.3 Hole Etching - $SF_6/O_2$ Plasma Etching . . . . . 53
6.4 Bosch Process - Deep Reactive Ion Etching . . . . . 57
6.5 DRAM Wiggling - $HBr/O_2$ Plasma Etching . . . . . 59
6.6 Profiling . . . . . 62
6.7 Energy Efficiency . . . . . 66

**7 Conclusion and Future Directions** **67**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# 1 Introduction

## 1.1 Motivation and Goal

Advances in semiconductor fabrication over the past several decades have made it one of the most important and strategically significant global industries. As processes get increasingly more complex and costly, simulation tools to test and research new ideas play a major role.

ViennaPS is a C++ library for topography simulation in microelectronic fabrication processes developed at the Institute for Microelectronics at TU Wien [1][2]. Key component of every simulated process step is the Monte-Carlo-based particle flux calculation on the given domain. The particles are represented by stochastic particle rays, where a single ray corresponds to a large number of physical particles. These rays are then approximated as straight lines traveling from a source to the surface of the domain without any particle-particle interactions. On the surface, they interact with the material at the impact location based on physical and chemical models. The computational cost to trace an individual particle ray and calculate the flux is relatively low. However, to achieve reasonable results, millions of rays have to be cast for every process iteration, while a full process can take thousands of iterations. The ray tracing component accounts for the vast majority of the computational time in each iteration of a process step. Since the goal is to achieve fast simulations without sacrificing accuracy, optimizing the ray tracing component of ViennaPS is expected to yield the greatest benefit.

When analyzing the problem, it becomes very clear that it is perfectly suited for GPUs, as the highly parallelized architecture is specifically designed for a large number of tasks where each task itself is cheap. To fully utilize the capabilities of modern GPUs, this implementation uses NVIDIA OptiX as a framework. It is designed and optimized for the given NVIDIA hardware and provides a comprehensive set of tools to implement high-performance ray tracing.

Prior to this work, ViennaPS was implemented with the option to execute the flux calculation on either the CPU, where the surface is approximated by tangential disks, or on the GPU, where it is composed of triangles. To provide more options and to study the effects of different geometry representations, this work implements three different GPU flux engines. One based on disks, similar to the CPU version. Another one based on lines, which are essentially the 2D counterpart to the 3D-only triangles. And finally, one directly using the ViennaPS internal structure to represent surfaces, the level-set.

## 1.2 Procedure and expected Outcome

The existing ViennaPS GPU triangle flux calculation engine will be expanded to also support disk, line, and level-set surface representations on the GPU. While there is already a CPU disk approach, porting the method to the GPU is expected to yield the same results while being noticeably quicker. The GPU line approach should be a two-dimensional counterpart to the three-dimensional triangle method, where similar effects should occur. The goal of using the level-set directly for ray tracing is to avoid the intermediate step of extracting an explicit mesh as this can introduce inaccuracies as well as additional computational overhead.
All flux engines will be benchmarked on various examples and compared in terms of total runtime,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

through visual inspection of the final surfaces as well as quantitative metrics of surface area and the Chamfer distance in relation to the reference solution produced by the CPU disk engine. The triangle engine already showed speedups of 20x-30x in previous benchmarks [3], so similar improvements are expected for the other engines as well.
The final implementations described here have been integrated into the ViennaPS and ViennaRay GitHub repositories where they are publicly available.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# 2 TCAD Simulations

## 2.1 Historical Background of TCAD

The term Technology Computer-Aided Design (TCAD) refers to a broad collection of simulation tools used to model and analyze semiconductor devices. It can be categorized in two main areas: process TCAD, which focuses on simulating fabrication processes and creating microscopic structures, and device TCAD, which extracts device properties from those resulting structures. The origins of TCAD can be traced back to the 1970s when one of the first simulators SUPREM-I was published by Stanford University in 1978 [4]. These pioneer simulators were already using continuum simulations and the drift-diffusion model which are very much still in use today. As the global demand for computational power increased, so did the demand for more complex semiconductor structures and, in turn, more sophisticated simulations. TCAD made advancements with the introduction of semi-classical and quantum mechanical models, allowing for more accurate predictions where some simulations take into account individual atoms in small scales.

Today, TCAD has to deal with multiple scales, bridging fabrication processes and final device behavior. This ranges from atomic scale simulations using molecular dynamics and density functional theory, to feature scale simulations using Monte-Carlo methods, all the way to simulating final devices by solving partial differential equations like Poisson's equation and the drift-diffusion model [5].

TCAD is one of the main drivers for innovations in that field as its ever-increasing accuracy and speed allows for faster research and development cycles and will probably stay relevant for as long as the semiconductor industry exists.

## 2.2 State of the Art Simulators

There are multiple commercial and academic TCAD tools offering solutions at one or more scales. Some of the most prominent vendors of commercial tools are:

- **Synopsys**: Offering a simulation suite with their Sentaurus product line [6].
- **Silvaco**: Offering similar tools with their Victory products [7].
- **Global TCAD Solutions (GTS)**: Providing the GTS Framework [8].
- **Lam Research**: With Semulator3D, a 3D process emulator that focuses on MEMS and advanced packaging technologies [9].

These tools are capable of simulating a wide range of process steps at multiple scales. They are usually designed to support the full TCAD workflow from start to finish by a single vendor. On the academic side there are also open-source options available. Some of them, specifically in the category of process TCAD, are:

- **ViennaPS**: A topography simulator developed at TU Wien focusing on fabrication processes using implicit surface representation and Monte-Carlo ray tracing for flux calculations [1][2].

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

- **Monte Carlo Feature Profile Model (MCFPM)**: Developed at the University of Michigan focusing on plasma etching processes [10].
- **SimProfile**: A surface profile simulator with automatic parameter calibration using experimental data [11].

All vendors and academic institutions have large interests in improving the runtimes of their tools. Typical simulations can be very time-consuming and take hours or even days to complete, while many iterations and variations are needed to make advancements in the field. As CPUs are reaching a limit in terms of clock speed and single-core performance, improvements in algorithms, parallelization, and hardware acceleration are needed to further accelerate simulations. In general, many simulations can take advantage of both shared memory parallelism using multi-core CPUs, and distributed memory parallelism using multiple nodes in a cluster. While it is always an option to use larger clusters with more nodes, it might not always lead to desired scaling due to communication overhead as well as being expensive in both setup and energy consumption.

Acceleration through dedicated hardware like GPUs has become increasingly popular in recent years. They can provide significant speedups for certain problems that can be parallelized effectively, which include solving large sparse linear systems as they appear from discretizing partial differential equations, or Monte-Carlo based ray tracing methods for flux calculations. Modern GPUs even offer specialized hardware on chip for tensor or ray tracing operations which can be utilized to further speed up simulations at all scales.

### 2.2.1 Atomic Scale Simulations

Atomic scale simulations focus on modeling materials and processes at the atomic or molecular level. This can range from calculating velocity distributions of particles inside a reactor to simulating quantum effects between neighboring atoms in a transistor. Techniques such as molecular dynamics (MD) and density functional theory (DFT) are commonly used in this domain [5]. MD simulations track the movement of atoms over time, making it possible to study phenomena like diffusion, phase transitions, and surface interactions. DFT, on the other hand, is a quantum mechanical method used to investigate the electronic structure of materials, providing insights into properties such as band structure and defect states. These simulations are computationally intensive and are feasible only for small systems (hundreds to a few thousand atoms for DFT and a few millions for MD). However, these methodologies provide detailed insights that are crucial for understanding fundamental material behaviors that influence larger-scale processes. They are also needed to calculate essential parameters for feature-scale simulations such as sticking probabilities or etch yields.

Popular academic software packages include LAMMPS for molecular dynamics and Quantum ESPRESSO, VASP, and CP2K for density functional theory calculations. An example of a commercial tool is Quantum ATK from Synopsys, which offers multiple atomistic models of varying complexity including newtonian and quantum mechanics, DFT, and semi-empirical models like Tight-Binding [12].

### 2.2.2 Feature Scale Simulations

Feature scale simulations deal with the modeling of fabrication processes at the scale of individual features, such as trenches and fins, on a semiconductor wafer, typically ranging from nanometers to micrometers. An example is Monte-Carlo based ray tracing to simulate particle deposition or etching. These simulations track the trajectories of particles (atoms, molecules, ions, etc.) that are initialized with characteristic values, such as velocity and energy, obtained from previous

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

atomic scale simulations or experiments. Upon impact with the surface, various interactions can occur, such as adsorption, reflection, or sputtering, depending on the particle's properties and the surface conditions. The cumulative effect of these interactions over many particles then leads to changes in the surface topography.

Feature scale simulations are essential for understanding how process parameters like gas flow and pressure translate to resulting device topographies and, in turn, varying device behavior. Furthermore, they are especially useful in testing out new materials and process steps without having to conduct expensive experiments.

ViennaPS is an example of an academic tool that falls into that category. There are also multiple commercial tools available like Silvaco Victory Process or Synopsys Sentaurus Process that offer feature scale fabrication simulations as part of their TCAD suites.

### 2.2.3 Device Scale Simulations

Device scale simulations focus on modeling the electrical behavior of semiconductor devices, such as transistors, diodes, and capacitors. Final device structures obtained from previous process TCAD serve as the input geometry for these simulations. This allows for analyzing how variations in fabrication processes impact device performance.

These simulations typically involve solving partial differential equations (PDEs) that describe the transport of charge carriers within the device. Relevant formulations include Poisson's equation for electrostatics and the drift-diffusion equations for carrier transport, which are combined to form the relatively old, but still widely used Drift-Diffusion-Model [13]. Numerical methods such as finite difference/element/volume methods (FDM/FEM/FVM) are employed to discretize and solve these equations over the device geometry. Another approach that also includes quantum-mechanical effects is the use of Non-Equilibrium Green's Function (NEGF) methods for simulating quantum transport in nanoscale devices [14].

Device scale simulations provide insights into performance metrics like current-voltage characteristics, switching speeds, and power consumption, giving engineers the ability experiment with new design or optimize existing ones.

Some tools are Synopsys Sentaurus Device, Silvaco Atlas, GTS Minimos-NT, and COMSOL Multiphysics. Depending on the scale and number of devices being simulated, there is a fluent transition to die scale simulations, often referred to as ECAD (electronic CAD), which can include millions or billions of devices. These simulations often rely on simplified compact models that approximate the behavior of individual elements to enable efficient circuit-level analysis [15].

## 2.3 Trends and Future Directions in TCAD

### 2.3.1 Hardware Accelerators

As an alternative to optimizing existing algorithms, a trend seen in many fields of computational science is to use specialized hardware to speed up simulations. Most commonly used are GPUs, which were originally designed for graphics rendering, but are now widely used for general-purpose computing (GPGPU) to accelerate parallelizable problems. Tensor Processing Units (TPUs) and Field-Programmable Gate Arrays (FPGAs) are other options though they are less common and lack the rich ecosystem that GPUs nowadays have.

Modern consumer-grade CPUs run on clock frequencies of 3–6 GHz supporting complex instruction sets with 8-16 cores and often double that amount in logical cores. Recent years have shown a plateau in CPU clock speeds and an asymptotic increase in single thread performance as shown in Fig. 2.1. The limiting factors are mainly heat generation, cost, and fabrication

TU Bibliothek, WIEN, Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

complexity. In the mid 2000s, CPU manufacturers shifted their focus from increasing clock speeds to adding more cores per CPU. Each core is very powerful but due to the capped core frequency and amount of cores, the overall throughput is limited.

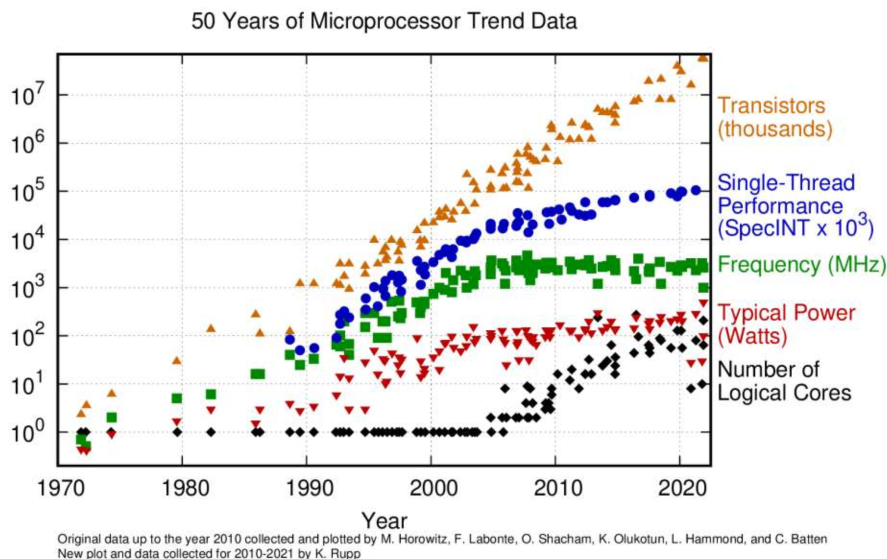



**Fig. 2.1:** Processor trends over the last 50 years. Source: Rupp [16]

GPUs, on the other hand, run on a lower clock speed while holding a larger number of cores. By lowering the clock speed, thermal issues are less of a problem and disproportionately more cores can be added while heat levels stay manageable. The current end-of-the-line consumer-grade GPU, the NVIDIA RTX 5090, released in Jan, 2025, holds more than 20,000 cores running at a frequency of 2–2.4 GHz [17]. While each individual core is significantly weaker compared to a CPU in terms of clock speed and instruction set, the raw amount of cores allows the GPU to process significantly more data than the CPU in the same time. GPUs are often used to speed up the solution of large sparse systems that arise from discretizing PDEs [18]. On top of that, modern GPUs usually have separate cores available that are specialized for ray tracing or matrix multiplication operations which can greatly enhance runtimes of computationally expensive segments of programs. In the context of TCAD, the matrix multiplication cores can be leveraged for accelerating machine learning models that are integrated into the simulation workflow, for process optimization or device performance prediction.

By utilizing these hardware accelerators, TCAD simulations can achieve significant performance improvements, allowing for more complex and detailed analyses to be conducted in shorter timeframes. Many, if not all, previously mentioned TCAD tools already support GPU acceleration to some extent. For example LAMMPS offers support for all major GPU vendors through OpenCL [19], while Quantum ESPRESSO originally supported only NVIDIA GPUs using CUDAFortran, but is currently under development to shift towards OpenACC and OpenMP5 to also target AMD and Intel GPUs [20]. Synopsys and COMSOL both focus on NVIDIA GPUs using CUDA.

### 2.3.2 Machine Learning in TCAD

Machine learning (ML) is becoming more and more relevant in TCAD simulations. This trend is tightly connected to the use of hardware acceleration, as training ML models requires significant

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

computational power and data. The models are used to predict fabrication process outcomes based on input parameters like process time, gas flow, pressure, temperature and initial geometry as well as predicting final device behavior [21]. Inference with an ML model could potentially output final geometry and physical properties faster than running an actual simulation. Additionally, these models could provide the option to slightly vary parameters to get a 'what-if' result, which could tremendously speed up the design process.

In order to achieve this, large amounts of data are needed for training. Ideally all this data would come from experiments, but relying on experimental data alone is too costly and slow. Synthetic data from simulations that is both in line with real world experimental data, and being multiple times cheaper and quicker to obtain is a promising alternative. This leads to a positive feedback loop where faster simulations lead to more data, which leads to faster and more accurate ML models that can then be used to speed up simulations even further.

## 2.4 Physical Process Simulation using ViennaPS

As mentioned previously, the focus of this work is on feature scale process simulations using ViennaPS. ViennaPS is a simulation tool that uses multiple dependencies, most importantly ViennaLS for implicit surface representations with level-sets, and ViennaRay for Monte-Carlo flux calculation on explicit surfaces. The connection between the dependencies can be seen in Fig. 2.2.

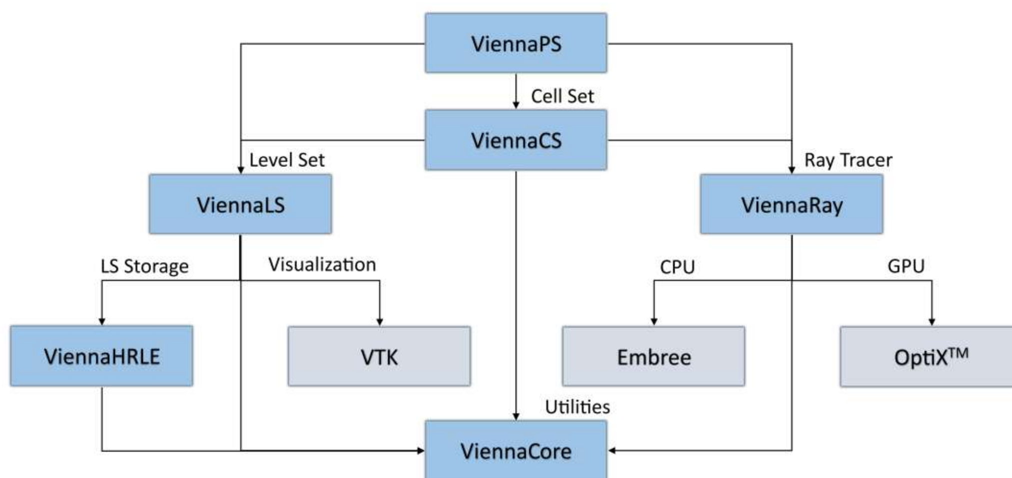



**Fig. 2.2:** ViennaPS framework. Source: Reiter & Filipovic [2]

The program flow for each time step is as follows: The simulation domain with its surfaces is defined in ViennaLS by a level-set grid, where each grid point holds a signed distance value and a normal vector. Only a sparse layer of grid points around the surface is stored using a hierarchical run length encoded (HRLE) data structure, which is provided by the ViennaHRLE library. This level-set grid is converted to an explicit surface that can then be used by the ray tracing engine in ViennaRay for the flux calculation, while ViennaPS provides the necessary physical context like particle energies and surface interactions. The current options for explicit meshes are a disk mesh, which can be used to run the ray tracing on the CPU, using Intel® Embree, or to convert it to a triangle mesh and perform the ray tracing on the GPU, using NVIDIA OptiX. The final fluxes are then used to calculate the surface advection velocities based on material and particle properties also defined in ViennaPS. Finally, these velocities are fed

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

back into ViennaLS where they are mapped from the explicit mesh elements back to the level-set grid. As a last step, ViennaLS advects the surface by updating the signed distance values using the calculated velocities. This is repeated for every time step of the process.

From previous benchmarks it is known that the majority of time spent by a process is during the flux calculation in ViennaRay, which is why the focus will be on optimizing this part of the program. It was already shown that using the GPU for this problem can lead to a massive improvement in runtime as depending on the specific process, considerably more rays can be traced in the same time than on the CPU [3].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

# 3 Fabrication Processes

This section provides the physical background for common fabrication processes used in semiconductor manufacturing. The focus will be on deposition and etching as they will be simulated later. A concise overview of the other process categories can be found in [22]. In general, processes can be divided into the following categories [23]:

- Deposition: A material is added to the substrate surface to form thin films or layers.
- Etching: Material is removed from the substrate surface by chemical or physical mechanisms.
- Lithography: A photoresist is used to coat the substrate. This photoresist can then selectively be hardened at specific areas using a mask by exposure to radiation. The unhardened areas can be removed using a solvent, while the hardened areas act as protective layers for subsequent processes, for example etching.
- Chemical Mechanical Planarization (CMP): A process used to smooth surfaces by removing material.
- Oxidation: A process that forms an oxide layer on the substrate surface. The oxide can act as an insulator or as a protective layer in subsequent processes.
- Ion Implantation: A technique to introduce dopants into the substrate by accelerating ions towards the surface.
- Diffusion: An alternative method to introduce dopants into the substrate by exposing it to a dopant-rich environment.

## 3.1 Material Deposition

One of the fundamental processes in semiconductor fabrication is material deposition, where material is added to the surface of a substrate. Common deposition techniques include Chemical Vapor Deposition (CVD), Physical Vapor Deposition (PVD), and Atomic Layer Deposition (ALD). Each method has its own advantages and disadvantages in terms of film quality and process complexity. The exact mechanisms of each method will not be discussed in detail here, but are well described in [24][25][26]. In general, deposition processes involve the transport of particles to the substrate surface, where they trigger chemical reactions or physically stick to the surface. The deposition rate and film properties can be influenced by temperature, pressure, gas flow rates, and substrate material. Fig. 3.1 shows the results of a simple deposition process depending on the particles sticking probability. An example would be the deposition of silicon dioxide ($SiO_2$) using tetraethyl orthosilicate (TEOS) as a precursor in a CVD process. The TEOS molecules are transported to the substrate surface, where they dissociate to form $SiO_2$ and volatile byproducts [27]. Deposition applications include:

- **Dielectric Layers**: Deposition of insulating layers such as silicon dioxide ($SiO_2$) and silicon nitride (SiNH) that can be used for capacitors, gate oxides, and interlayer dielectrics [27].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Passivation Layers**: Deposition of protective coatings to prevent etching of underlying layers in future processing steps.
- **Metal Films**: Deposition of conductive materials like aluminum and copper for interconnects and contacts in integrated circuits.
- **Barrier Layers**: Deposition of materials that prevent diffusion of unwanted species, such as titanium nitride (TiN) used as a barrier between silicon and aluminum [28].

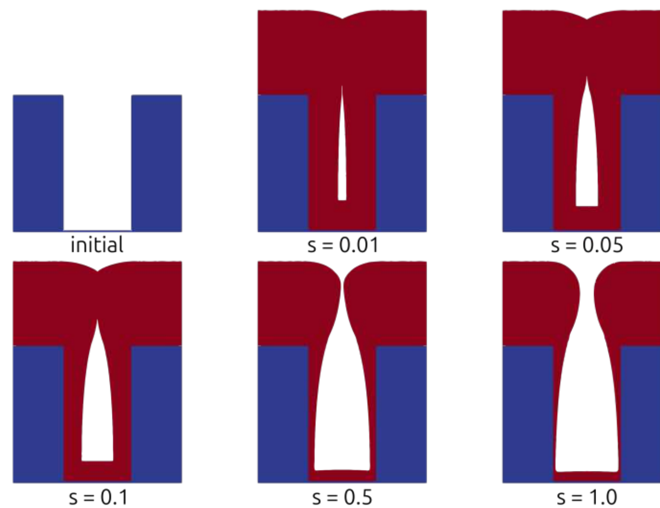



**Fig. 3.1:** Schematic illustration of particle deposition on a substrate surface with varying sticking probabilities. Source: ViennaPS [29]

## 3.2 Etching

Etching is the process of removing material from the substrate surface to create desired patterns and structures. There are two primary types of etching: wet etching and dry etching. Wet etching involves the use of liquid chemicals to dissolve material, while dry etching utilizes plasma or reactive gases to remove material through physical or chemical means [22]. There are various etching techniques with varying final profile characteristics. With increasing complexity of semiconductor devices and features becoming smaller and smaller, it became clear that high-aspect-ratio (HAR) structures which take advantage of the vertical dimension are necessary, which require advanced, highly anisotropic etching methods. Some applications of HAR structures include:

- **FinFETs**: Fin Field-Effect Transistors (FinFETs), as seen in Fig. 3.2, are a type of non-planar transistor architecture that uses a vertical fin-like structure to increase the effective channel width. The fins are created using HAR etching techniques to achieve the necessary aspect ratios [30].
- **Through-Silicon Vias (TSVs)**: TSVs are vertical electrical connections that pass through silicon wafers or dies through deep and narrow holes to enable 3D integration of semiconductor devices [31].
- **Micro-Electro-Mechanical Systems (MEMS)**: MEMS devices often require HAR structures. Etching techniques like deep reactive ion etching (DRIE) are commonly used to fabricate these features [32].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

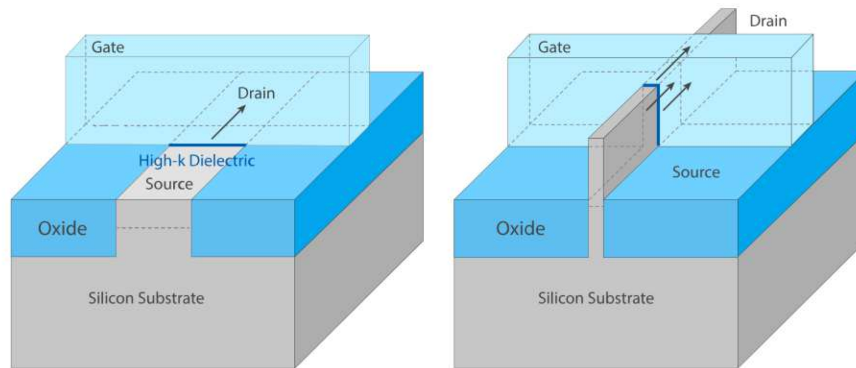


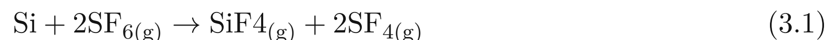

**Fig. 3.2:** A planar transistor compared to a FinFET structure showing the increased effective channel width in FinFETs due to the vertical fin. Source: Faraday Technology [33]

### 3.2.1 $SF_6/O_2$ Plasma Etching

One advanced and widely used etching technique capable of achieving relatively high aspect ratios is $SF_6/O_2$ plasma etching. The total etch rate is influenced by three effects:

- **Chemical etching** by the reaction of fluorine radicals with silicon.
- **Physical sputtering** by energetic ions from the plasma accelerated towards the substrate.
- **Ion-enhanced etching** by a combination of the two. Silicon that is covered with fluorine has a lower binding energy and is more easily removed by incoming ions.

In a plasma, $SF_6$ dissociates into fluorine radicals and various sulfur-fluorine fragments. The fluorine radicals are highly reactive and interact with the silicon surface to form volatile silicon fluorides that can be pumped away, effectively removing silicon material.



$$Si + 2SF_{6(g)} \rightarrow SiF4_{(g)} + 2SF_{4(g)} \tag{3.1}$$

Physical sputtering occurs when energetic ions from the plasma are accelerated towards the substrate surface and transfer their momentum to surface atoms, causing them to be ejected if the ions kinetic energy $E_{ion}$ is higher than the surface binding energy $E_{th}$.
Ion-enhanced etching is a synergistic effect where the presence of fluorine on the surface lowers the binding energy of silicon atoms, making them more susceptible to removal by incoming ions. The surface can be covered by fluorine, which accelerates the etching process, or by oxygen, which forms a passivation layer that protects the surface from etching.

By balancing the concentrations of $SF_6$ and $O_2$ in the plasma, it is possible to control the etch profile. A higher $SF_6$ concentration leads to more fluorine radicals and thus more isotropic etching, while more $O_2$ passivates the sidewalls leading to anisotropic profiles, demonstrated in Fig. 3.3 [34].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

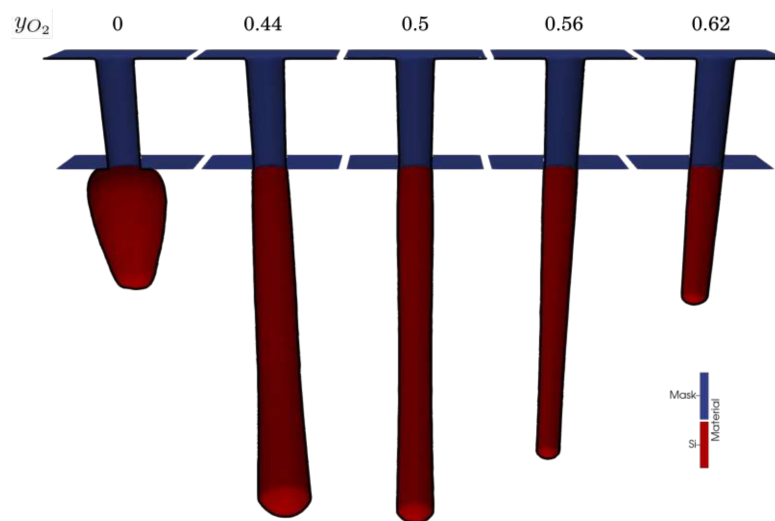



**Fig. 3.3:** Simulation results of $SF_6/O_2$ plasma etching with varying concentrations of oxygen. Source: ViennaPS [29]

### 3.2.2 Bosch Process

Another common etching technique is deep reactive ion etching (DRIE), specifically the Bosch process. It involves alternating cycles of plasma etching and deposition of passivating polymer layers to achieve deep, vertical sidewalls. During the etching phase, silicon is removed through chemical reactions with fluorine radicals and physical sputtering by accelerated ions from the plasma. The chemical etching is isotropic, meaning that it removes material in all directions equally. This leads to horizontal etching which is undesirable when trying to create vertical structures. After the etch phase, a passivation phase follows, where a polymer layer is deposited on the surface. This layer protects the sidewalls from further etching in the next etch phase. In the second etch phase, the ions, which are moving mostly vertically, interact more with the bottom of the crater where they sputter off the passivation layer and expose the underlying silicon to the chemical etchant, while the sidewalls remain protected. This cyclic process allows for control over the etch profile and depth and results in characteristic scalloped sidewalls, illustrated in Fig. 3.4 [35].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

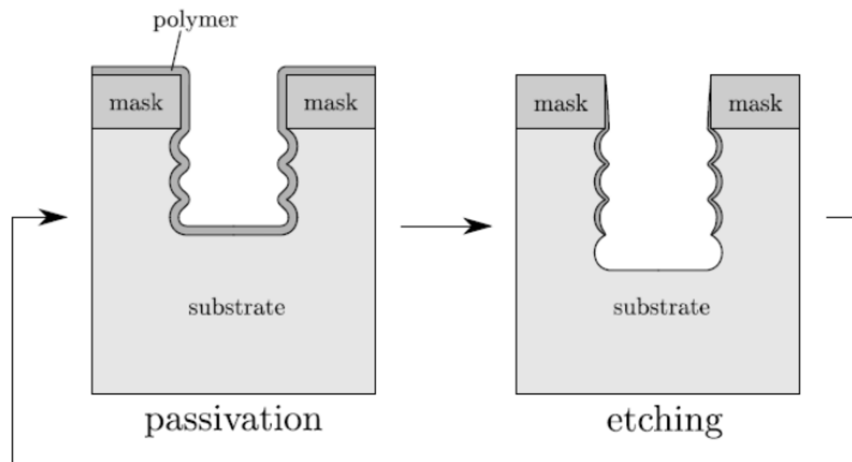



**Fig. 3.4:** Schematic illustration of the Bosch process showing alternating etching and passivation steps resulting in vertical scalloped sidewalls. Source: Ertl [35]

### 3.2.3 $HBr/O_2$ Plasma Etching

$HBr/O_2$ plasma etching is another technique very similar to $SF_6/O_2$ plasma etching used for anisotropic etching of silicon [36]. In fact, they can even be used together in a single gas mixture as analyzed in [37]. In this process, HBr provides bromine radicals instead of the previous fluorine radicals, while $O_2$ still acts as a passivation layer on the sidewalls. One application is the fabrication of Dynamic Random Access Memory (DRAM).

One of the critical components of DRAM is the trench capacitor, which stores electrical charge in deep trenches etched into the silicon substrate. These trenches need to be very deep and narrow to maximize the yield of memory cells and memory density on a given silicon wafer [38]. A common problem during the etching of these trenches is a phenomenon called DRAM Wiggling. It refers to the sidewalls of the trenches exhibiting a wavy profile instead of being perfectly straight.

During plasma etching, byproducts are created that can stick to the sidewalls of the trench. These byproducts form a passivation layer that protects the sidewalls from further etching. In general, this passivation layer is directly removed by vertically moving ions that sputter off the layer at the bottom of the trench. However, as the fins are spaced closely together, some areas require more silicon to be removed than others to achieve a uniform trench depth. In Fig. 3.5 site A requires more etchant than site B due to the shifted pattern layout, which means that site A also builds up more byproducts which passivate the sidewalls. This causes less etching at site A compared to site B, which makes the fin at site A bulging outwards, while site B stays vertical. This deformation can have significant impact on the electrical properties like capacitance and leakage currents of the final device. Accurately simulating this effect is crucial for optimizing fabrication processes and improving device reliability [39].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

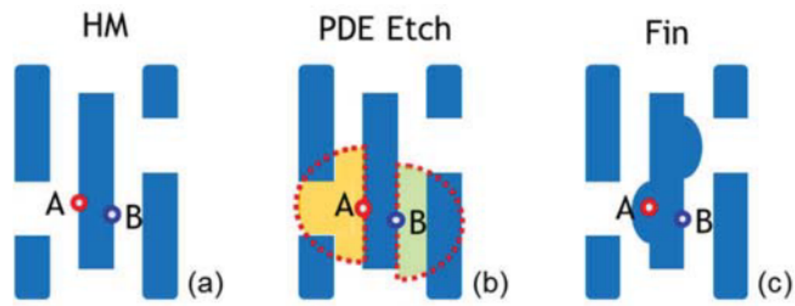



**Fig. 3.5:** Illustration of AA profile in Fin etch process (a) hard mask top view before etch, (b) pattern dependence etch amount comparison between site A and B, (c) top view after fin etch. Source: Wang et al. [39]

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

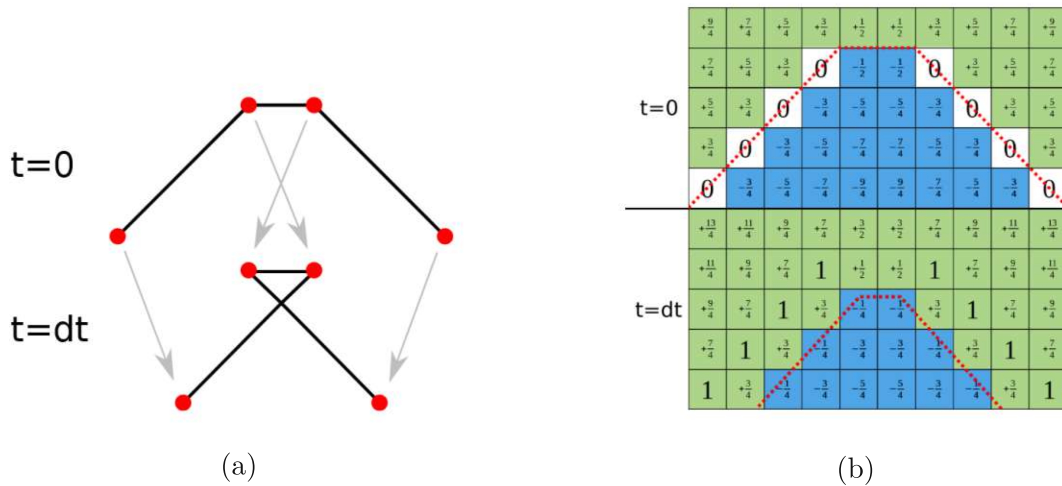
# 4 Process Simulation

## 4.1 Level-Set Surface Representation

In a traditional sense, geometry in computer graphics or simulations is usually represented using explicit meshes made up of vertices, edges, and faces. While this representation is very intuitive and easy to visualize, it has some drawbacks when it comes to simulating processes where the shape of the geometry can change. During processes like deposition and etching, surfaces can split up or merge together, which is difficult to handle with explicit meshes and comes with an additional computational cost.

ViennaPS uses an implicit surface representation based on a signed distance function (SDF) $\Phi(\vec{x})$ to represent geometry, also called a level-set. An SDF is a scalar field defined over a two- or three-dimensional grid where each point in the grid stores the shortest distance to the nearest surface. The sign of that value indicates whether the point is inside (negative) or outside (positive) the geometry. The surface itself is the zero level-set of the SDF, meaning all points where the signed distance is zero.

This approach has several advantages. Firstly, the resolution of the geometry always stays the same, no matter how it is deformed. It can also be easily adjusted by changing the grid spacing. Secondly, topological changes like merging, splitting, or overlaps, demonstrated in Fig. 4.1, are handled intuitively without any special treatment.




(a) (b)

**Fig. 4.1:** Comparison of surface evolution with an explicit and implicit surface. (a) The edges in the explicit surface overlap after moving the vertices which would require additional handling. (b) The implicit surface naturally handles topological changes without additional effort. Source: Klemenschits et al. [40]

To move a surface during a time step, each value in the grid is updated based on a local velocity $\vec{v}$ in the normal direction of the surface. Mathematically, this is described by the level-set equation which is a Hamilton-Jacobi equation, often solved using finite difference methods [40]:

$$\frac{\partial \Phi(\vec{x},t)}{\partial t} + H(\vec{x}, \nabla\Phi, t) = 0 \tag{4.1}$$

$$\text{with} \quad H(\vec{x}, \nabla\Phi, t) = v(\vec{x})|\nabla\Phi(\vec{x},t)| \tag{4.2}$$

By separating the terms, multiplying by $\partial t$ and integrating over a time step we get

$$\int_t^{t+\Delta t} \partial\Phi(\vec{x},t)\partial t = -\int_t^{t+\Delta t} H(\vec{x}, \nabla\Phi, t)\partial t, \tag{4.3}$$

which results in the update equation for each grid point and time step

$$\Phi(\vec{x}, t+\Delta t) = \Phi(\vec{x},t) - \int_t^{t+\Delta t} H(\vec{x}, \nabla\Phi, t)\partial t. \tag{4.4}$$

ViennaPS features multiple finite difference schemes to solve this equation with trade-offs between accuracy and computational cost. The simplest method is first-order Engquist-Osher, also known as the upwind scheme. It is computationally inexpensive but only stable for convex Hamiltonians. It uses one-sided differences to approximate the spatial derivatives based on the direction of the velocity. Higher order Engquist-Osher schemes use more grid points to approximate the derivatives and achieve higher accuracy at the cost of computational effort.

$$\int_t^{t+\Delta t} H(\vec{x}, \nabla\Phi, t)\partial t = \Delta t \cdot v(\vec{x}) \begin{cases} \frac{\Phi(\vec{x}+\vec{e}_i) - \Phi(\vec{x})}{\Delta x} \\ \frac{\Phi(\vec{x}) - \Phi(\vec{x}-\vec{e}_i)}{\Delta x} \end{cases} \tag{4.5}$$

Higher accuracy and stability can be achieved using the Lax-Friedrichs scheme which uses central differences to approximate the solution.

$$\int_t^{t+\Delta t} H(\vec{x}, \nabla\Phi, t)\partial t = \Delta t \cdot v\nabla^0\Phi(\vec{x}) - \Delta t \sum_{i=1}^{D} \alpha_i \cdot \frac{\partial_i^+\Phi(\vec{x}) - \partial_i^-\Phi(\vec{x})}{2} \tag{4.6}$$

$$\nabla^0\Phi(\vec{x}) := \sqrt{\sum_{i=1}^{D} \left(\frac{\partial_i^+\Phi(\vec{x}) + \partial_i^-\Phi(\vec{x})}{2}\right)^2} \tag{4.7}$$

$$\alpha_i \geq \max_x \left|\frac{\partial H}{\partial q_i}\right| \quad \text{with} \quad q_i := \frac{\partial \Phi}{\partial x_i}. \tag{4.8}$$

Independent of the actual scheme used, the time step $\Delta t$ has to be chosen carefully to ensure numerical stability. This is performed by satisfying the Courant-Friedrichs-Lewy (CFL) condition which limits the time step based on the grid spacing $\Delta x$ and the maximum velocity in the domain with a factor $C_{\text{CFL}}$, which must be less than 1:

$$\Delta t = C_{\text{CFL}} \cdot \frac{\Delta x}{\max_x |v(\vec{x})|} \tag{4.9}$$

ViennaPS, or more specifically its underlying library ViennaLS, uses a value of 0.5 for the $C_{\text{CFL}}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

## 4.2 Particle-Particle Interaction

All fabrication processes are typically carried out in specialized reactors under controlled conditions of temperature, pressure, and gas flow. The choice of process parameters significantly affects the quality and characteristics of the resulting structures. The simulation of the conditions inside the reactor chamber itself is outside the scope of this work. Given the environmental conditions in a reactor chamber, and assuming the particles behave like an ideal gas, we can calculate the mean free path $\lambda$. The mean free path is a measure of the average distance a particle travels before colliding with another particle. It is given by the temperature $T$, pressure $p$, Boltzmann constant $k_B$ and the effective diameter of the particles $d$

$$\lambda = \frac{k_B T}{\sqrt{2}\pi d^2 p}, \tag{4.10}$$

where typical values range from $T$=300-1000 K, $p = 1 - 10^5\ Pa$ and $d = 0.4\ nm$ depending on the specific process and gas used. This results in the mean free path being in the scale of micrometers. Since the features we want to simulate are typically in the scale of nanometers to a few micrometers, the mean free path is often larger than the feature dimensions. This implies that particles are unlikely to collide with each other within the feature scale, allowing us to neglect particle-particle collisions in our simulations. By ignoring these interactions, we can simplify the simulation model by treating particles as independent entities that are traveling in a straight line from source to surface. This allows us to focus on particle-surface interactions, which are more important for accurately modeling deposition and etching processes at the feature scale.

## 4.3 Source Distribution

The simulation starts by creating particles just above the wafer surface. The position of each particle is randomly sampled from a uniform distribution over the source plane. The directions are sampled from a (powered) cosine distribution following Lambert's cosine law. The exact shape of the distribution depends on results from reactor-scale simulations and can be calibrated to match experimental data. Charged particles may be accelerated in an electric field towards the substrate, resulting in a more directional flux. This can be modeled by adjusting the exponent of the cosine distribution, which is illustrated in Fig. 4.2 for neutral and charged particles [40].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

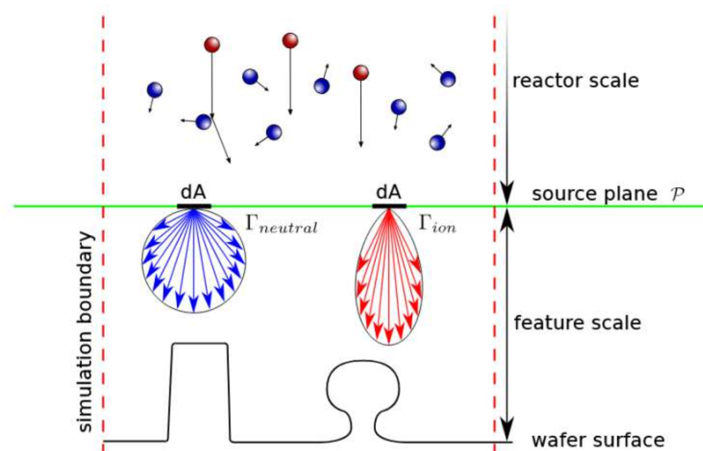



**Fig. 4.2:** Schematic illustration of the cosine distribution used to sample particle directions at the source. Source: Klemenschits et al. [40]

## 4.4 Particle-Surface Interactions

When particles reach the surface of the substrate, they can undergo various interactions depending on their energy, angle of incidence, and the material properties of the surface. Common interactions include adsorption, reflection, sputtering, and chemical reactions. Physically, a particle either sticks to the surface and contributes to the interaction, or it is reflected. From a simulation perspective, this can be modeled in different ways. One option would be to directly implement the physical behavior and treat each particle as an individual entity that sticks to a surface based on a probability between 0 and 1. A comparison with a random number from a uniform distribution then decides whether the particle sticks or is reflected.

Another approach, which is used in ViennaPS, is to model this behavior using weighted particles. Each particle starts off with a weight of 1, representing the probability of it still being active. At each interaction with the surface, the weight is added to a surface element specific counter after which the weight is reduced by the sticking probability. If the weight falls below a certain threshold, the particle is terminated. This approach has the advantage that a single particle is a statistical entity that can contribute to the final flux at multiple locations in correct proportion. With this approach, the final fluxes should be distributed more smoothly while having to launch fewer particles in total, thus reducing computational cost.

The sticking probability is not always a constant value but can depend on multiple factors such as the particle's energy, angle of incidence, the material, and current state of the surface (e.g., concentration of absorbed chemicals). All these factors have to be taken into account and calibrated to experimental data [22].

### 4.4.1 Simple Deposition/Etching Models

The simplest model for particle-surface interaction is to treat all particles independent of each other without any synergetic effects. This means that each particle interacts with the surface based on its own properties only, without being influenced by previous interactions or the current state of the surface. This approach is suitable for processes where the interactions are primarily physical and do not involve complex chemical reactions. Examples include physical vapor deposition (PVD) or sputtering processes. The model can simulate both neutral and charged particles [40].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

#### 4.4.1.1 Neutral Particles

Simulating neutral particles is relatively straightforward. When hitting the surface, the current weight is added to the element specific flux counter, its weight is reduced by the sticking probability, and the particle is diffusely reflected from the surface using a cosine distribution around the normal. This continues until the weight falls below a certain threshold and the particle is terminated. The sticking probability is a constant value that can be set for the particle itself or for each surface element individually based on the surface material [40].

#### 4.4.1.2 Charged Particles

Simulating ions can also be simple if the energy of the ions is not considered. In this case they behave similar to neutral particles, but their sticking probability is a linear falloff of the angle of incidence between a minimum and maximum angle

$$S_{\text{ion}} = \begin{cases} 1 & \text{if } \theta \leq \theta_{\text{Rmin}}, \\ 1 - \dfrac{\theta - \theta_{\text{Rmin}}}{\theta_{\text{Rmax}} - \theta_{\text{Rmin}}} & \text{if } \theta > \theta_{\text{Rmin}}, \\ 0 & \text{if } \theta \geq \theta_{\text{Rmax}}. \end{cases} \tag{4.11}$$

The other difference to neutral particles is that the reflection direction is not diffuse, but in a cone around the specular reflection direction with a certain spread angle, as seen in Fig. 4.3. If the energy of the ions is set to a non-zero value, the interaction becomes more complex and the flux results from multiplying the particle weight with an energy and angle dependent yield factor $Y_{sp}$. The ions can contribute to the total flux by sputtering only if their kinetic energy is above a threshold. The total sputtering yield then results by multiplying with $f(\theta)$ which specifies the angular dependence.

$$Y_{sp}(E, \theta) = Y(E) \cdot f(\theta) \tag{4.12}$$

$$Y(E) = \begin{cases} 0 & \text{if } E < E_{th} \\ \sqrt{E} - \sqrt{E_{th}} & \text{if } E \geq E_{th} \end{cases} \tag{4.13}$$

$$f(\theta) = (1 + B_{sp}(1 - \cos^2(\theta))) \cos(\theta) \tag{4.14}$$

After the collision, the energy of the reflected ion is reduced based on the angle of incidence and a calibration factor $n$

$$E_{\text{ref}} = \begin{cases} 1 - (1 - A) \dfrac{\frac{\pi}{2} - \theta}{\frac{\pi}{2} - \theta_{\text{inflect}}} & \text{if } \theta \geq \theta_{\text{inflect}}, \\ A \left( \frac{\theta}{\theta_{\text{inflect}}} \right)^{n_l} & \text{if } \theta < \theta_{\text{inflect}}, \end{cases} \tag{4.15}$$

where $A = \left(1 + n\left(\frac{\pi}{2\theta_{\text{inflect}}} - 1\right)\right)^{-1}$ [40].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

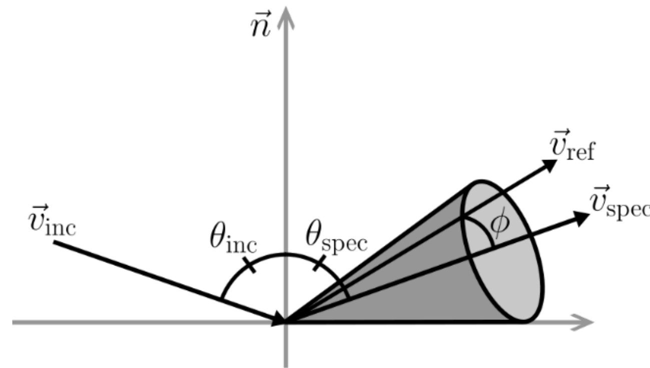



**Fig. 4.3:** Schematic illustration of ion reflection showing the specular direction and reflection cone. Source: Klemenschits et al. [40]

### 4.4.2 Plasma Etching Models

The following description is based on the example of $SF_6/O_2$ plasma etching. The general principles can be transferred to other plasma chemistries like $HBr/O_2$ with some adjustments to the constant parameters [23].

Simulating plasma etching is more complex than simple deposition or etching models due to the synergetic effects between different species in the plasma and their interactions with the surface. Fluorine radicals and oxygen are simulated as neutral particles, while the ions from the plasma are treated as charged particles. In general, these behave very similar to the descriptions above in terms of sticking probability, reflection angle, and energy after reflection, but the model has to take into account the synergetic effect of ion-enhanced etching.

To simulate this, each surface element has a coverage value that represents the presence of fluorine $\theta_F$ and oxygen $\theta_O$ with a probability. These coverages are kept in balance by Langmuir-Hinshelwood-type surface site balance equations, given by

$$\sigma_{Si}\frac{d\theta_F}{dt} = \beta_F\Gamma_F(1-\theta_F-\theta_O) - k\sigma_{Si}\theta_F - 2Y_{ie}\Gamma_i\theta_F, \tag{4.16}$$

$$\sigma_{Si}\frac{d\theta_O}{dt} = \beta_O\Gamma_O(1-\theta_F-\theta_O) - \beta\sigma_{Si}\theta_O - Y_O\Gamma_i\theta_O, \tag{4.17}$$

where $\sigma_{Si}$ is the density of silicon atoms on the surface, $\beta_{F/O}$ are the sticking coefficients, $\Gamma_{F/O/i}$ the source fluxes of fluorine, oxygen and ions, $k$ and $\beta$ are chemical constants and $Y_{ie/O}$ are the ion-enhanced and oxygen etching yields.

As the incoming flux is in the range of $10^{16} - 10^{19}\text{cm}^{-1}\text{s}^{-1}$ and is therefore orders of magnitude larger than the surface change rate, which is in the order of $10^{-9}\text{ms}^{-1}$, a quasi-steady state assumption can be made and the time derivatives set to zero. This results in closed equations that can be used to get the current coverage value at each surface element once the incoming flux of particles $\gamma$ at each element is known.

$$\theta_F = \left[1 + \left(\frac{k\sigma_{Si} + 2Y_{ie}\Gamma_i}{\gamma_F\Gamma_F}\right)\left(1 + \frac{\gamma_O\Gamma_O}{\beta\sigma_{Si} + Y_O\Gamma_i}\right)\right]^{-1} \tag{4.18}$$

$$\theta_O = \left[1 + \left(\frac{\beta\sigma_{Si} + Y_O\Gamma_i}{\gamma_O\Gamma_O}\right)\left(1 + \frac{\gamma_F\Gamma_F}{k\sigma_{Si} + 2Y_{ie}\Gamma_i}\right)\right]^{-1} \tag{4.19}$$

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The coverages have an impact on the sticking probabilities of the neutral particles, since a particle is only able to stick to the portion of the surface which is not already covered by another particle.

$$S_{\mathrm{F/O}} = S_{F/O}(1 - \theta_F - \theta_O) \tag{4.20}$$

The final etch rate is then given by a combination of chemical etching, physical sputtering and ion-enhanced etching.

$$\mathrm{ER} = \frac{1}{\rho_{Si}} \left( \frac{k\sigma_{Si}\theta_F}{4} + Y_{sp}\Gamma_i + Y_{ie}\Gamma_i\theta_F \right) \tag{4.21}$$

The sputtering yield stays the same as in Eq. (4.12), while for the ion-enhanced sputtering a lower threshold energy and for the angular dependence $f(\theta) = \cos(\theta)$ is used. Also, a yield coefficient $A$ is introduced to calibrate the model [34][41]:

$$Y_{sp}(E, \theta) = A_{sp} Y(E) f(\theta) \tag{4.22}$$

$$Y_{ie}(E, \theta) = A_{ie} Y(E) \cos\theta \tag{4.23}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

# 5 Implementation

## 5.1 GPU-accelerated Ray Tracing Frameworks

To efficiently simulate particles going from source to surface at the feature scale, we leverage GPU-accelerated ray tracing frameworks to avoid having to write an entire ray tracing engine from scratch. These frameworks offer high level functionality to simplify the development process while also being well optimized.

With NVIDIA's introduction of the Turing architecture in 2018 came the first hardware-accelerated implementation of ray tracing using separate RT cores. AMD followed up with the RDNA 2 architecture in 2020 and their Ray Accelerators (RA), as well as Intel with their Alchemist architecture in 2022 and Ray-Tracing Units (RTU). Regardless of vendor, ray tracing cores are specifically designed for ray traversal in a scene and ray-triangle intersection, and while a direct comparison to 'regular' GPU cores is not straightforward, they are several times faster in their specific task. NVIDIA claims that the NVIDIA RTX 2080 Ti can trace 10x more rays per second than its predecessor, the NVIDIA GTX 1080 Ti, mostly because of RT cores [42].

These ray tracing cores are generally not directly accessible from regular compute APIs like CUDA or OpenCL but can be utilized through higher level frameworks for which a variety of options exists. Each has its own advantages and disadvantages, some of the most notable ones include:

- **Intel Embree/SYCL**: Originally designed to run on CPUs only, a GPU support via SYCL was added in 2023. SYCL can only access hardware accelerators on Intel hardware.
- **NVIDIA OptiX**: Benefits from the large CUDA ecosystem, proprietary but free to use for most applications. Widely used in research and industry for high-performance ray tracing applications. Limited to NVIDIA GPUs.
- **DirectX Raytracing**: Windows only, proprietary. It is mainly used for real-time rendering in video games. Good support for all major GPU vendors.
- **Vulkan RT**: Cross platform, open standard. Also, mainly used for real-time rendering and has good support for all major GPU vendors.

ViennaRay uses Intel Embree for CPU-based ray tracing and NVIDIA OptiX for utilizing the GPU. As Embree only supports hardware accelerators on Intel GPUs, which are neither competitive in terms of performance, nor as widely adopted compared to NVIDIA products at the moment, and the GPU support being relatively new, this option was not considered further. NVIDIA OptiX was chosen for this implementation due to its maturity and extensive CUDA ecosystem. Also, NVIDIA GPUs are the most commonly used GPUs in research and high performance computing. DirectX Raytracing with its focus on real-time rendering in video games and other interactive applications is not specifically suited for this use case. It is also Windows only which would limit the accessibility of the software. The next alternative would be Vulkan RT, which also supports hardware accelerators provided by all GPU vendors and is an open standard. However, it is not as widely used in research and is also meant for applications in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the video game industry. It might however be the best option to support all major GPU vendors while also utilizing ray tracing cores.

To be completely vendor-agnostic, open source, and free of any design limitations, one could consider using OpenCL which is a low-level API that is not specifically designed for ray tracing but general compute workloads. While it is very flexible and can run on almost any hardware, much of the functionality required for efficient ray tracing would need to be implemented from scratch, which is a significant commitment. Also, OpenCL does not have access to the ray tracing specific hardware accelerators. This limitation alone could drastically hurt performance and make it a weaker choice than Vulkan RT.

## 5.2 NVIDIA® OptiX™

OptiX provides functionality to build acceleration structures for fast ray-geometry traversal, as well as built-in ray-triangle intersection tests, which are both executed on the RT cores. Since it is specifically designed for NVIDIA GPUs, it is highly optimized for the given architecture and fully utilizes the hardware capabilities.

OptiX also allows for the use of custom primitives as surface elements, but in this case only the ray traversal is performed by the RT cores while the intersection tests are handled by regular CUDA cores. This means that the custom primitives in this implementation will also benefit from the hardware acceleration.

### 5.2.1 Bounding Volume Hierarchy Acceleration Structure

The brute-force approach for ray tracing would be to check every ray against every geometric primitive in the scene. With increasing ray count and geometry complexity this quickly becomes computationally unfeasible. The key to fast ray tracing is efficient ray traversal to quickly find the primitives that may be intersected, while ignoring others, which are guaranteed to not be hit. To achieve this, the geometry is organized in a spatial acceleration structure that groups primitives that are close to each other. One common approach that is used in OptiX [43], as well as in Intel Embree [44], is a Bounding Volume Hierarchy (BVH) acceleration structure, which is schematically shown in Fig. 5.1.

A BVH is a tree structure on a set of geometric primitives. Each node in the tree represents a bounding volume that contains a subset of mesh elements. The leaf nodes contain the actual primitives that make up the geometry. If a ray does not intersect a bounding volume, all primitives contained within that volume can be skipped. This significantly reduces the number of intersection tests required, leading to faster runtimes [45].

Very importantly, the BVH traversal in OptiX does not guarantee any specific order, meaning that the first leaf node that was hit by the ray is not necessarily the closest one to the ray's origin. This will be important when later trying to find the closest intersection and also any intersections in proximity to that one.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

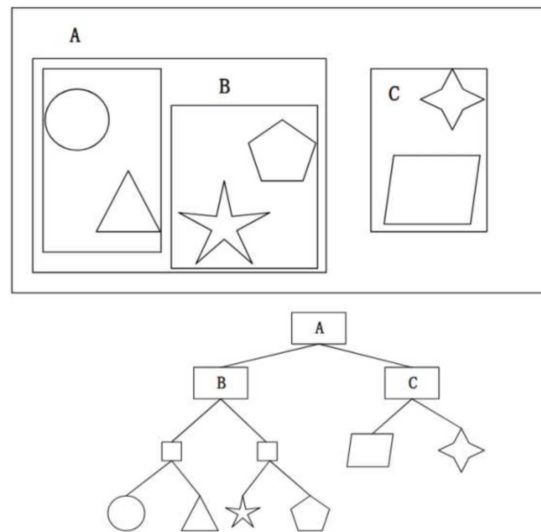



**Fig. 5.1:** Schematic example of a Bounding Volume Hierarchy: The root node A contains the whole scene. C has two leaf nodes containing the actual geometry. B holds two internal child nodes, each with two leaf nodes. Source: Hu et al. [46]

### 5.2.2 Shader Binding Table and Pipeline Programs

The Shader Binding Table (SBT) is an array that holds information about the location of programs and their parameters used in the ray tracing pipeline. It acts as a lookup table, so each thread that is tracing a ray knows which program to execute when a ray hits a primitive. In this implementation it is also used to store the whole mesh data in GPU memory, so that it is accessible from within the pipeline programs. The SBT is the central piece that links geometry, materials, and programs together into a single pipeline. A ray tracing pipeline in OptiX consists of several programs, their relations are shown in Fig. 5.2 [43]:

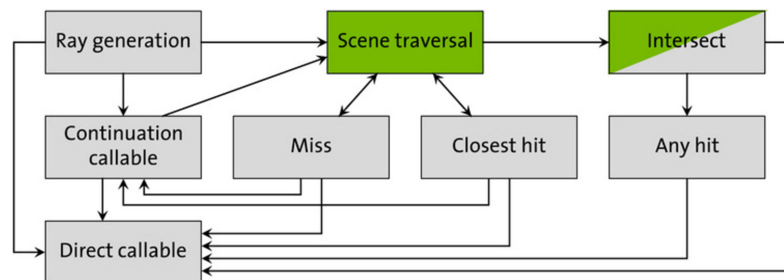



**Fig. 5.2:** OptiX Pipeline Program relations. Source: NVIDIA [43]

- Ray Generation (RG): This program is responsible for initializing rays with an origin and direction. It starts the ray tracing process and begins with the BVH traversal. It is the only program that is always required in every pipeline.
- Intersection (IS): Called when the BVH traversal finds a leaf node that holds a primitive that is potentially hit. It runs the intersection test and if an intersection is found, it may report that hit which can then be processed by the two Hit-Programs. When an intersection is reported, OptiX will not search for any other intersections that are guaranteed to be farther away than the reported one to optimize traversal [47]. When using triangles, curves

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

or spheres as primitives, no IS program is required as the built-in one is used. For custom geometry types, a custom IS program has to be provided.

- Any Hit (AH): Called when a reported intersection is potentially closer than the current closest intersection. It is important to understand that this program will not be executed for every potential intersection as again once the IS program reported an intersection it will never report any intersections that are farther away.
- Closest Hit (CH): Called for the intersection that is closest to the ray's origin once the BVH traversal cannot find any other potential hits. This program is typically used to compute effects at the surface and reflections.
- Miss (MI): Executed when a ray does not intersect any geometry in the scene. In most cases the ray is terminated.
- Direct callable (DC): Act like functions that can be called from within any of the above programs in the pipeline and are executed directly. They are especially useful as alternatives to function pointers for avoiding inlining large portions of code or code duplication.
- Continuation callable (CC): Similar to direct callables but are executed by the scheduler. Also, they can recursively launch new rays. Can only be called from RG, CH, and MI.
- Exception (EX): Handles errors that may occur during the execution of the pipeline.

The pipelines in this implementation consist of a RG, IS, CH, and MI program combined with direct callables. The flowchart in Fig. 5.3 shows the execution order of the different programs in the pipeline.

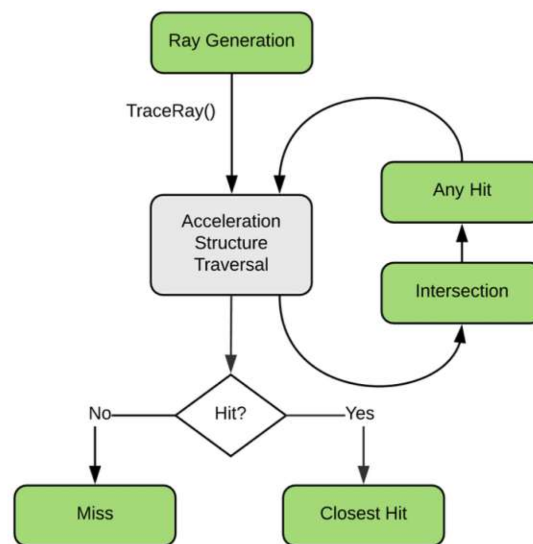



**Fig. 5.3:** OptiX Flowchart. Source: NVIDIA [48]

## 5.3 Meshing and Axis-Aligned Bounding Boxes

ViennaPS uses ViennaLS to represent surfaces as level-sets. ViennaLS uses ViennaHRLE internally, which is a sparse grid data structure to store the signed distance function (SDF)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

values and normals of the level-set. Grid points that are close to the actual surface are marked as active, while inactive points are ignored in order to have a minimal memory footprint.

To perform ray tracing on the surface, the ViennaHRLE data structure needs to be translated to something OptiX understands. OptiX gives the freedom of using any custom primitive to describe surfaces, it only expects an axis-aligned bounding box (AABB) around each element for building the BVH structure. The AABBs need to be large enough to fully contain the underlying primitive ensuring no potential hits are missed, while being as small as possible to keep the BVH traversal efficient. Using the optimal AABB for each geometry type is therefore crucial for correctness and performance.

OptiX also does not provide any built-in boundary conditions for the simulation domain. Boundary elements which are set at the faces of the domain volume have to be provided. The boundary elements are one large primitive (or two primitives in the case of triangles) per face that fully enclose the simulation domain. With that, it is possible to detect if a ray hit a boundary element and implement custom boundary conditions. ViennaRay supports both periodic and reflective boundaries.

The disk and level-set mesh are available in both 2D and 3D, while the line and triangle mesh are only available in 2D or 3D respectively. In 2D, the mesh and simulation is still in three-dimensional space with all z-coordinates set to zero.

### 5.3.1 Disk Mesh

The most efficient mesh extraction is to shift all active points in their normal direction by the distance given by their signed distance value. This results in a point cloud where each point can be interpreted as a disk with a certain radius, demonstrated in Fig. 5.4 [49]. The radius should be large enough to not leave any gaps in the surface, but small enough to not unnecessarily overlap too much with neighboring disks, as this would lead to more intersection tests and more neighbor smoothing. The disk radius $r$ is set to the distance between two grid points $\Delta$ multiplied by half the length of the diagonal of a unit square or unit cube, depending on whether the mesh is two- or three-dimensional, respectively.

$$\mathrm{r} = \begin{cases} \Delta \cdot \frac{\sqrt{2}}{2} & \text{for 2D} \\ \Delta \cdot \frac{\sqrt{3}}{2} & \text{for 3D} \end{cases} \tag{5.1}$$

To define an optimal AABB around a disk, one can use the disk center $\vec{c}$, normal $\vec{n}$, and radius $r$ to find the extent of the box in each dimension from the center point. An orthonormal basis, consisting of the normal $\vec{n}$ and two perpendicular vectors $\vec{u}$ and $\vec{v}$ that span the disk plane, is defined. The components of these basis vectors must fulfill the Pythagorean theorem.

$$n_i^2 + u_i^2 + v_i^2 = 1 \quad \text{for } i = x, y, z \tag{5.2}$$

Any point on the edge of the disk can be reached by a combination of vectors $\vec{u}$ and $\vec{v}$. The edge is given by

$$r \cdot \sqrt{u_i^2 + v_i^2} = r \cdot \sqrt{1 - n_i^2}. \tag{5.3}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Therefore, the AABB can be defined as

$$\text{AABB}_{\text{min,max}} = \vec{c} \mp r \cdot \begin{pmatrix} \sqrt{1-n_x^2} \\ \sqrt{1-n_y^2} \\ \sqrt{1-n_z^2} \end{pmatrix}. \tag{5.4}$$

The boundary elements are one large disk per boundary face. The center point is the center of the face, the normal is pointing inwards and the radius is the largest extent of the simulation domain in one dimension multiplied by 0.5 in 2D and $\frac{\sqrt{3}}{2}$ in 3D. While this guarantees that the boundary disks fully enclose the simulation domain, this can also lead to disproportionately large disks that overlap with other boundary disks if the domain is not cubic. This is not a problem for correct ray tracing results, but should be adapted in the future when optimizing for runtime.

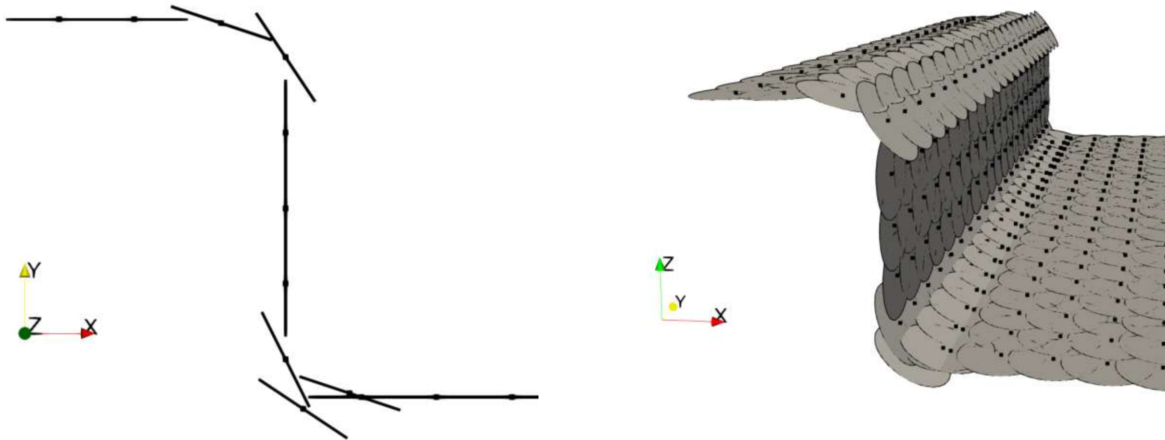



**Fig. 5.4:** Tangential disk meshes in 2D and 3D.

### 5.3.2 Line/Triangle Mesh

ViennaLS uses the marching squares/cubes algorithm to extract a line mesh in 2D and a triangle mesh in 3D. An example mesh can be seen in Fig. 5.5. The elements are stored as vertices and a pair or a triplet of indices respectively. As triangles are a built-in type in OptiX, no AABBs have to be provided for them. To define an optimal AABB around a line segment, one can simply use the two endpoints $\vec{a}$ and $\vec{b}$.

$$\text{AABB}_{\text{min}} = \min(\vec{a}, \vec{b}) \tag{5.5}$$

$$\text{AABB}_{\text{max}} = \max(\vec{a}, \vec{b}) \tag{5.6}$$

In 2D, the boundary elements are one large line per boundary edge. The endpoints are the two corners of the edge. In 3D, the boundary elements are two large triangles per boundary face. The vertices are the four corners of the face split into two triangles. The normals are again pointing inwards for both mesh types.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

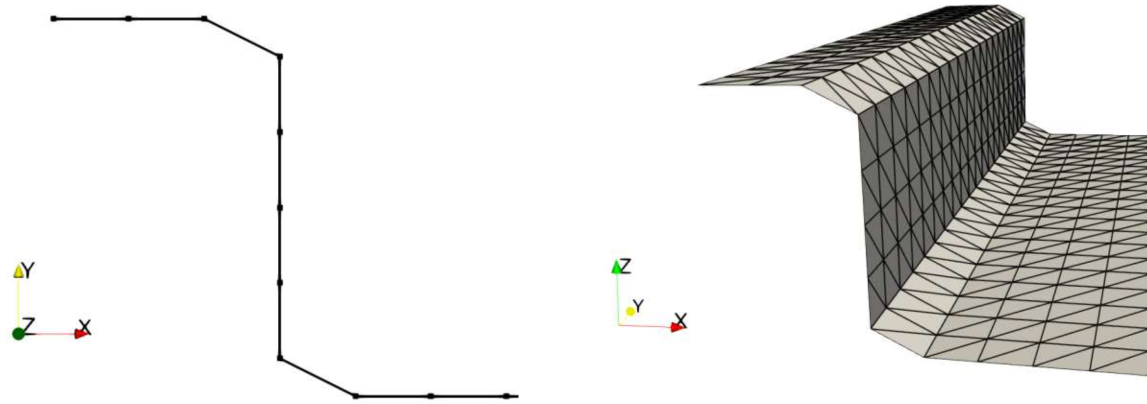



**Fig. 5.5:** Line and triangle meshes in 2D and 3D.

### 5.3.3 Level-Set Mesh

As extracting a mesh from the level-set can lead to loss of detail and also be computationally expensive, using the level-set stored in the ViennaHRLE structure directly for ray tracing would be preferable. Since OptiX still requires AABBs around the smallest primitive that makes up the geometry, we cannot provide the level-set as a whole. Instead, we have to divide it into cubic voxels, where each voxel extends by one grid spacing in each dimension. This is illustrated in Fig. 5.6. The corners of a voxel hold the SDF values from the level-set grid. Each voxel has a unique three-dimensional index that can be used to find the position of the minimal corner of the voxel in space, by multiplying the index with the grid delta. In 2D, the SDF values are copied and extruded in the z-direction to artificially create a 3D voxel. This is not optimal in terms of memory usage and algorithmic complexity as it creates a lot of redundant data, but it simplifies the implementation significantly. The AABBs are then trivially defined by the position of the voxel's minimum corner $\vec{v}$ and the grid delta $\Delta$.

$$\mathrm{AABB}_{\mathrm{min}} = \vec{v} \cdot \Delta \tag{5.7}$$

$$\mathrm{AABB}_{\mathrm{max}} = \mathrm{AABB}_{\mathrm{min}} + \Delta \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{5.8}$$

The boundary elements are one large voxel per boundary face. The voxels are placed so the center of the face aligns with the center of the voxel. All boundary voxels are cubes with an edge length of $\Delta_b$, which is determined by the largest extent of the simulation domain in one dimension. Similar to the disk mesh boundaries, this can lead to voxels that are disproportionately large while also overlapping with other boundary voxels. This should be addressed in the future when optimizing. The SDF values are set so that the inward pointing faces hold a value of 0 while the outward pointing faces hold a value of -$\Delta_b$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

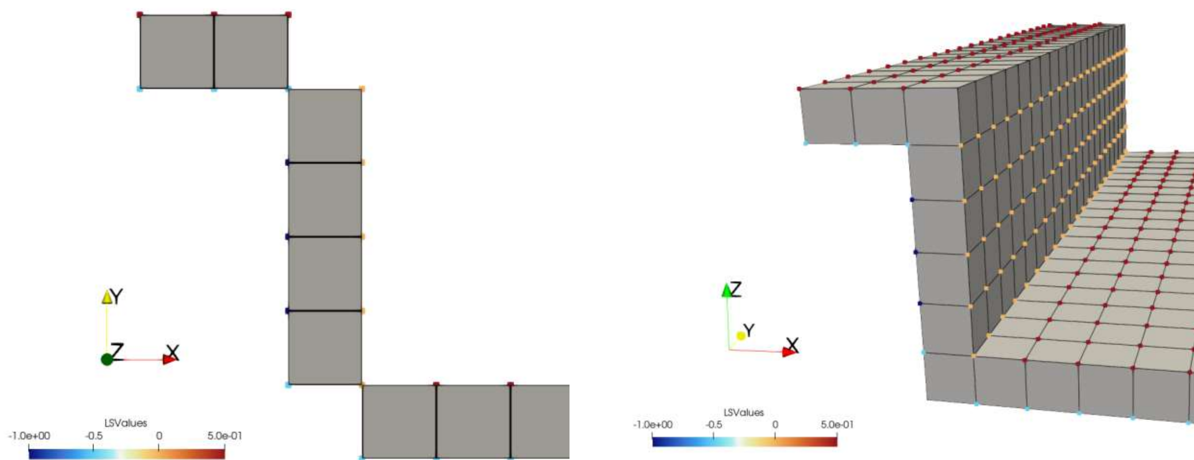



**Fig. 5.6:** Voxel meshes with a level-set grid in 2D and 3D.

## 5.4 Ray-Generation (RG)

The ray tracing process starts by generating rays in the RG program of the OptiX pipeline. A ray is always traced from creation to termination by a single thread. All information about a ray is stored in a structure called `PerRayData` (PRD) which is accessible from every program within the pipeline during the ray's lifetime. When initializing the PRD, the origin and direction are sampled randomly. The origin is pulled from a uniform distribution over the source plane, which spans the entire simulation domain in the XY-plane in 3D and the X-axis in 2D, immediately above the highest surface element. The direction is sampled from a cosine-weighted hemisphere from the source with -Z and -Y as general directions in 3D and 2D.

All the CPU and GPU versions use an implementation of a Philox RNG to generate random numbers. Since the implementations are not identical, the exact same ray origins and directions cannot be guaranteed. This can lead to small differences in the final result, which should not be problematic due to the large number of rays. This will be discussed when comparing the disk method on the CPU and GPU, as they should produce identical results given the same ray configurations.

The PRD also holds information relevant for identifying and storing neighboring surface elements, which will be discussed in the next section. The size of the PRD structure should be kept minimal to avoid unnecessary memory reads and writes during ray traversal, which can decrease performance. In the current implementation, all flux engines use the same PRD structure for compatibility reasons even though the line and triangle engines do not require all fields. Having those unnecessary fields in the PRD leads to a slight performance overhead of 2-3% for those engines, which could be avoided at the cost of additional code.

All rays start off with a weight of 1, which is then decreased during ray-surface interactions (CH) depending on the sticking probability of the particle hitting the surface. If a ray hits a surface element, the weight is reduced and a reflection angle is calculated using the elements normal. By using the intersection location and reflection direction, the ray's origin, direction, and weight in the PRD are updated. Once the weight drops below a threshold, the ray is terminated.

ViennaRay has the option to launch a fixed number of rays, or to launch a certain number of rays per surface element. A rule of thumb is to launch 1.000-3.000 rays per surface element to get reasonable results.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

## 5.5 Ray-Element-Intersection (IS)

This section describes the intersection tests for the different geometry types used in the IS program of the OptiX pipeline. They are executed for each primitive that is contained in the leaf node of the BVH, that was hit during traversal. A ray $\vec{p}$ is defined by its origin $\vec{o}$ and direction $\vec{d}$:

$$\vec{p} = \vec{o} + t \cdot \vec{d}, \quad t \in [0; \infty) \tag{5.9}$$

In general the goal is to find a value $t_i$ at which the ray intersects the surface element. If such a value exists, the intersection point $p_i$ is then given by

$$\vec{p_i} = \vec{o} + t_i \cdot \vec{d}. \tag{5.10}$$

If there is no intersection, the Miss program (MI) is called, which will terminate the ray.

### 5.5.1 Ray-Disk Intersection

A disk $D$ is defined by its center point $\vec{c}$, normal $\vec{n}$ and radius $r$:

$$D = \left\{ \vec{p} \in \mathbb{R}^3 \;\middle|\; (\vec{p} - \vec{c}) \cdot \vec{n} = 0 \;\text{ and }\; \|\vec{p} - \vec{c}\| \leq r \right\} \tag{5.11}$$

To find the intersection with a disk, we first have to find the intersection with the disk's plane. For that we insert the equation for the ray into the equation of the plane.

$$(\vec{o} + t \cdot \vec{d} - \vec{c}) \cdot \vec{n} = 0 \tag{5.12}$$

Then, solving for $t_i$ gives

$$t_i = \frac{(\vec{c} - \vec{o}) \cdot \vec{n}}{\vec{d} \cdot \vec{n}}. \tag{5.13}$$

To decide if the ray hit the disk, the distance from center to intersection point has to be smaller than the radius of the disk. To avoid calculating a costly square root we can also compare the squared distance to the squared radius.

$$\|\vec{p_i} - \vec{c}\| \leq r \;\text{ or }\; \|\vec{p_i} - \vec{c}\|^2 \leq r^2 \tag{5.14}$$

#### 5.5.1.1 Overlapping Disk Hits

As disks inherently overlap in our representation, to avoid leaving any gaps, it is possible that a ray intersects multiple neighboring disks. To avoid a discontinuous flux on the surface, each intersection has to be registered and counted. In theory, we want to store $t_i$ for every intersection of the ray with the geometry, find the smallest one $t_{min}$ and count every hit that lies in the interval $t_i \in [t_{min}, t_{min} + \Delta t]$ with a given threshold $\Delta t$. OptiX does not guarantee any order in which the intersections are found, as well as ignoring any BVH nodes that are farther away than a previously reported intersection. Finding every intersection is still possible, but is a waste of resources as the pruning of far away nodes is a crucial traversal optimization. There are multiple ways to handle this.

The CPU version takes a simple approach. During mesh extraction, neighborhood information containing the IDs of neighbors is stored for every disk. A disk is defined as a neighbor if the distance of the center points is smaller than two times the disk radius. With that criterion the neighbors are almost always just the first order neighbors. First the closest hit is found, then the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

intersection test is explicitly rerun with the same ray for every neighbor. In general this method works well; however, it does not reuse information from the BVH traversal, which might have already shown that some neighbors are definitely (not) hit. Also, depending on the neighbor radius, potential intersections could be missed in some cases, as demonstrated in Fig. 5.7.

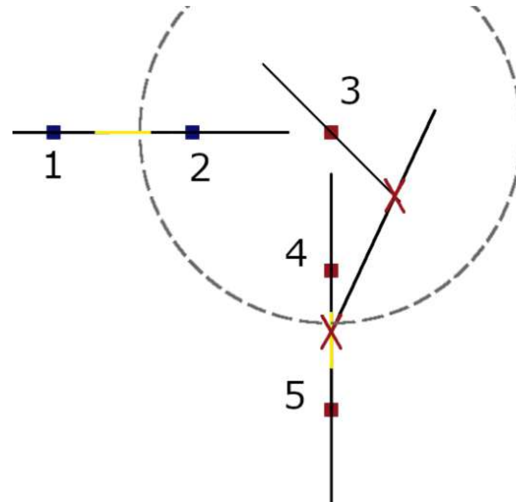



**Fig. 5.7:** The ray hits disk 3 and then proceeds to hit a spot where disks 4 and 5 overlap (marked yellow). In the CPU version, only disk 4 would be registered as a neighbor hit as the center of disk 5 is not within the neighborhood radius.

If a ray hits a disk at a corner in a steep angle, it is possible that the second intersection is at a spot where two disks overlap. If we only consider first order neighbors for our intersection tests, we might miss a disk that would have the same $t_i$ value for the intersection as a neighbor disk, which does get hit. This will lead to lower flux for elements that are second order neighbors of corner disks. The problem can be solved by increasing the neighbor radius; however, by increasing the number of neighbors we also drastically increase the number of intersection tests that have to be performed. In this simple 2D example in Fig. 5.7, including second order neighbors will increase the amount of neighbor intersection tests from 2 to 4, doubling the workload. In 3D this effect becomes even more pronounced. This results in a significant waste of resources, especially since the BVH traversal might have already ruled out some unnecessary tests.

The GPU disk version takes a different approach. In addition to reporting every found intersection in IS, the $t$ value as well as the ID of the hit primitive is manually stored in the PRD. If we were to report the correct $t_i$ value and OptiX finds the closest hit first, it will prune all other hits, including the ones we are interested in. To avoid this, we report every $t$ value plus a desired threshold $\Delta t$. If OptiX now finds the closest hit first, it will see that the intersection was at $t_i + \Delta t$ and therefore still search for other intersections that are within our interval. By doing this, we can ensure that all intersections are found that are within the threshold of the closest hit, while keeping the BVH traversal optimized and the number of intersection tests low. By manually storing the hit neighbors, we also do not have to explicitly rerun the intersection tests, saving more resources.

This method also rules out the special case at corners described in Fig. 5.7, as we are not bound to a fixed number of neighbors, but rather a range of $t$ values. Smaller values for $\Delta t$ decrease runtime, while potentially missing more neighbor hits, whereas larger values increase the number of intersection tests and accuracy. The threshold value must not be too large as otherwise a ray may hit two elements that are obviously not neighbors but within the threshold

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

range, an example for this is found in Fig. 5.8. A value of 1.1 times the grid delta was found to be a good compromise between accuracy and performance.

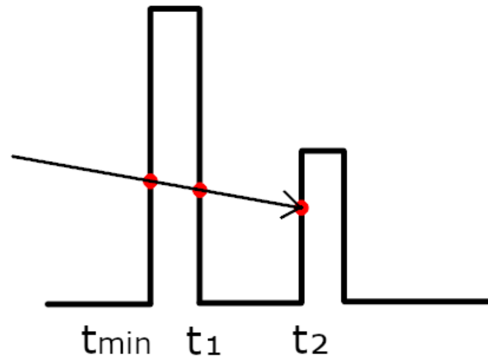



**Fig. 5.8:** A too large $\Delta t$ could lead to counting $t_2$ as a hit, although this is geometrically impossible. $t_1$ can be filtered by comparing the normals of the ray and the element.

### 5.5.2 Ray-Line Intersection

A line segment $L$ is defined by its two endpoints $\vec{a}$ and $\vec{b}$ where $\vec{l}$ is the connecting vector of those points:

$$L = \left\{ \vec{p} \in \mathbb{R}^2 \,\middle|\, \vec{p} = \vec{a} + u \cdot \vec{l},\ u \in [0;1] \right\} \tag{5.15}$$

To find an intersection we have to find $t$ and $u$ such that

$$\vec{o} + t \cdot \vec{d} = \vec{a} + u \cdot \vec{l} \text{ for } t > 0. \tag{5.16}$$

By applying the cross product of $\vec{l}$ to both sides we get

$$(\vec{o} + t \cdot \vec{d}) \times \vec{l} = (\vec{a} + u \cdot \vec{l}) \times \vec{l}. \tag{5.17}$$

Since the cross product of a vector with itself is zero we can solve for $t_i$.

$$t_i = \frac{(\vec{a} - \vec{o}) \times \vec{l}}{\vec{d} \times \vec{l}} \tag{5.18}$$

Similarly, by applying the cross product of $\vec{d}$ to both sides we also get $u_i$.

$$u_i = \frac{(\vec{o} - \vec{a}) \times \vec{d}}{\vec{l} \times \vec{d}} \tag{5.19}$$

To decide if the ray intersects the line segment, the following conditions must be met:

$$t_i \geq 0,\ u_i \in [0;1] \tag{5.20}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The normal at the intersection point is trivially given by swapping and negating the components of the direction vector while keeping track of the orientation.

$$\vec{n} = \frac{1}{||\vec{l}||} \begin{pmatrix} -l_y \\ l_x \end{pmatrix} \tag{5.21}$$

### 5.5.3 Ray-Level-Set Intersection

A level-set surface is defined as the zero-level of a signed distance function $\phi : \mathbb{R}^3 \to \mathbb{R}$, where $\phi(\vec{p})$ gives the shortest distance from point $\vec{p}$ to the surface, with negative values being inside the surface and positive values outside.

$$S = \left\{ \vec{p} \in \mathbb{R}^3 \;\middle|\; \phi(\vec{p}) = 0 \right\} \tag{5.22}$$

ViennaLS provides a grid that holds the signed distance values at each grid point. By using linear interpolation in each dimension, it is possible to interpolate the signed distance at any point in space. For simplicity, Fig. 5.9 shows a single 2D cell in the level-set grid defined by its four corner points $\vec{c}_{00}, \vec{c}_{10}, \vec{c}_{01}, \vec{c}_{11}$ with their corresponding signed distance values $\phi_{00}, \phi_{10}, \phi_{01}, \phi_{11}$.

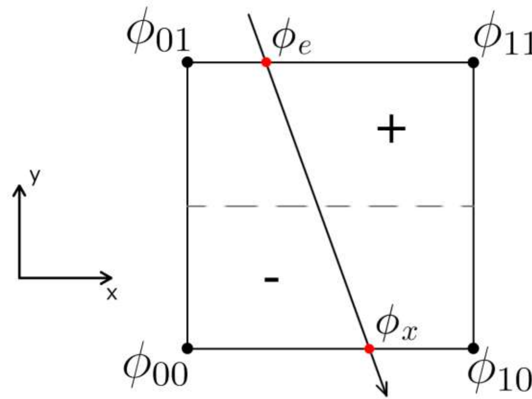



**Fig. 5.9:** 2D cell of a level-set grid with signed distance values $\phi_{ij}$ at the corners. The ray enters the cell at $t_e$ and exits at $t_x$. The signed distance values at those points are $\phi_e$ and $\phi_x$.

We first find the entry and exit points of the ray with the cell's bounding box and linearly interpolate the signed distance values at those points. If $\phi_e$ and $\phi_x$ have different signs, the ray intersects the level-set within this cell. The value for $t_i$ can then be found by linear interpolation along the ray between the entry and exit points.

$$t_i = t_e - \frac{\phi_e}{\phi_x - \phi_e}(t_x - t_e) \tag{5.23}$$

This linear interpolation is an approximation of the actual intersection point as it assumes the signed distance function varies linearly along the ray within the cell. This is not true for when the implicit surface is curved. To take the curvature into account, trilinear interpolation along the ray would be needed, which results in third order polynomials [50][35][51]. Such a more complex method was also tested, but it had little to no impact on the final result. Since the resulting fluxes will be averaged with their neighbors anyway, sub-grid accuracy is not necessary. This implementation will use the simpler linear interpolation for now.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The surface normal at the intersection point is found using central differences of the interpolated signed distance function in a small neighborhood $\epsilon$ around the intersection point.

$$\vec{n} = \nabla\phi(\vec{p}_i) = \frac{1}{2\epsilon}\begin{pmatrix}\phi(\vec{p}_i + \epsilon\vec{e}_x) - \phi(\vec{p}_i - \epsilon\vec{e}_x)\\ \phi(\vec{p}_i + \epsilon\vec{e}_y) - \phi(\vec{p}_i - \epsilon\vec{e}_y)\\ \phi(\vec{p}_i + \epsilon\vec{e}_z) - \phi(\vec{p}_i - \epsilon\vec{e}_z)\end{pmatrix} \tag{5.24}$$

#### 5.5.3.1 Ray-Box-Intersection

Even though we defined an AABB around each voxel, OptiX does not provide a function that returns the entry and exit points of a ray with that box; they must be calculated manually. To do so, the slab method was used. In each dimension there is a slab between the two parallel faces of the voxel that extend infinitely. The intersection of all three slabs is the volume of the voxel. The goal is to find all intervals for $t$, for which the ray is inside each slab. If there is a common interval for all three then the ray hit the voxel. The voxel spans from $\vec{l}$ to $\vec{h}$ in each dimension. The slabs are defined as

$$S_i = \left\{\vec{p} \in \mathbb{R}^3 \;\middle|\; p_i \in [l_i; h_i]\right\} \text{ for } i = x, y, z. \tag{5.25}$$

By inserting the ray equation into the slab equation we can find the intervals for $t$ in each dimension.

$$t_i^{low} = \frac{l_i - o_i}{d_i} \tag{5.26}$$

$$t_i^{high} = \frac{h_i - o_i}{d_i} \tag{5.27}$$

$$t_i^{min} = \min(t_i^{low}, t_i^{high}) \tag{5.28}$$

$$t_i^{max} = \max(t_i^{low}, t_i^{high}) \tag{5.29}$$

The intersection of all these segments is

$$t^{min} = \max(t_x^{min}, t_y^{min}, t_z^{min}) \tag{5.30}$$

$$t^{max} = \min(t_x^{max}, t_y^{max}, t_z^{max}). \tag{5.31}$$

If $t^{min} < t^{max}$ and $t^{min} > 0$, then the ray hit the voxel. The entry point is at $t_e = t^{min}$ and the exit point at $t_x = t^{max}$ [52]. A 2D example of the slab method is shown in Fig. 5.10.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

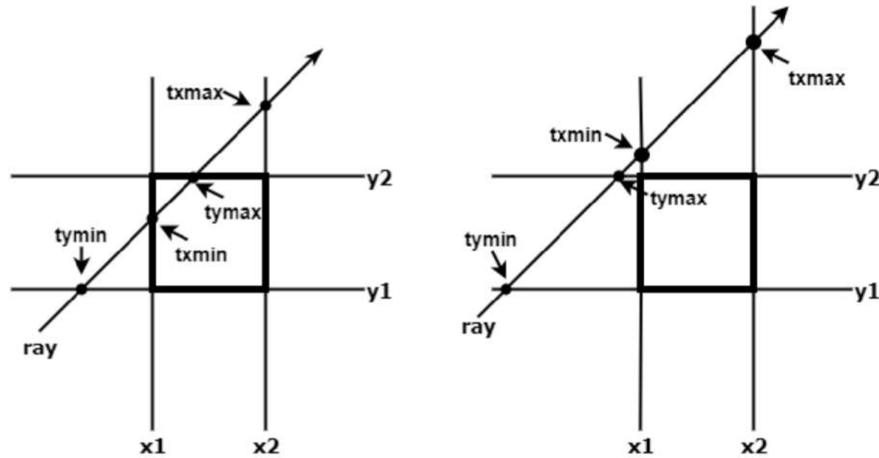



**Fig. 5.10:** Left: $t_x$ and $t_y$ share a common interval from $t \in [t_x^{min}, t_y^{max}]$. Right: there is no common interval, the ray misses the box. Source: Chatzianastasiou & Constantinides [52]

#### 5.5.3.2 Linear Interpolation within a Voxel

To find the SDF value at any point inside the voxel, linear interpolation of the SDF values at the corners was used. First the coordinates of the point of interest $\vec{x}$, in our case the entry and exit points, are converted to local voxel coordinates $\vec{x}_l$, where $\vec{x}_0$ is the global coordinate of the minimal corner of the voxel.

$$\vec{x}_l = \frac{1}{\Delta}(\vec{x} - \vec{x}_0) \tag{5.32}$$

We can find the trilinear interpolation by first interpolating the faces in x direction, the edges in y direction, and the corners in z direction. By abbreviating the linear interpolation as

$$\text{lerp(a,b,c)} = a + (b - a)c, \tag{5.33}$$

we get the SDF value at point $\vec{x}$ as

$$\begin{aligned} c_{00} &= \text{lerp}(\phi_{000}, \phi_{100}, x_l) \\ c_{01} &= \text{lerp}(\phi_{010}, \phi_{110}, x_l) \\ c_{10} &= \text{lerp}(\phi_{001}, \phi_{101}, x_l) \\ c_{11} &= \text{lerp}(\phi_{011}, \phi_{111}, x_l) \\ c_0 &= \text{lerp}(c_{00}, c_{01}, y_l) \\ c_1 &= \text{lerp}(c_{10}, c_{11}, y_l) \\ \phi(\vec{x}) &= \text{lerp}(c_0, c_1, z_l) \end{aligned} \tag{5.34}$$

[53]. With the interpolated SDF values at the entry and exit points of the ray with the voxel, we can find the intersection point through linear interpolation along the ray as described in Eq. (5.23).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

## 5.6 Ray-Surface-Interaction (CH)

Once the BVH traversal is complete, all intersection tests have been performed, and the closest hit has been determined, the ray can interact with the surface. Depending on the physical model, the particle type, and surface material, different interactions are possible. All these interactions are defined in the Closest-Hit program (CH) and the physical models provided by ViennaPS. Generally, if a ray hits a surface element, the ray weight is added to a hit counter associated with that element, after that the ray weight is reduced by a sticking probability and the ray is reflected. This was already described in Section 4.4 so this section will focus on implementation details and optimizations.

### 5.6.1 Boundary Hits

One check that has to be performed in the CH program is if the closest hit was a boundary element. If so, the ray is either reflected or wrapped around to the other side of the domain depending on the selected boundary condition. For reflective boundaries, we calculate the reflection direction by specular reflection. By using reflective boundaries and taking advantage of symmetry, it can be possible to cut the simulation domain in half or even in quarters which can significantly save computation time as the number of surface elements is reduced. Every time a boundary is hit, a ray-specific boundary hit counter in the `PerRayData` is incremented by one. If this counter exceeds a threshold of 100, the ray is terminated to prevent infinite loops.

### 5.6.2 Backside Hits

During a simulation, millions of rays are traced, some of which could end up in specific edge cases, where floating point errors and programming inaccuracies could lead to rays passing through the surface and ending up inside the material. From there, they could hit the backsides of surface elements and cause incorrect simulation results. To prevent this, the ray direction is compared with the normal of the surface primitive that was hit. If the angle between the two is larger than 90 degrees, the ray hit the backside of the element. In general if the closest hit was on a backside the ray is terminated; however, for the disk mesh this requires special handling.

A ray might hit the backside of a disk even though it is on the correct side of the surface. This can happen due to disks sticking out of the surface, as seen in Fig. 5.11, or diffuse reflections which make the ray move nearly parallel to the surface. If a backside is hit, that information is stored as a flag in the `PerRayData` and the ray is let through once without counting the erroneous hit. As this should be quite rare, if a second backside hit occurs there is probably something wrong with the ray, and it is terminated.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

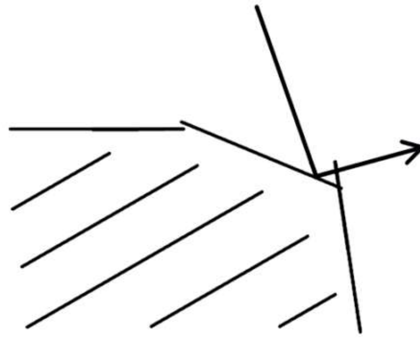


**Fig. 5.11:** The ray is reflected and hits the backside of a neighbor disk.

### 5.6.3 Direct Callables (DC)

After ruling out boundary and backside hits, the actual ray-surface interaction can be performed. The interaction depends on the particle type as well as the material of the surface element. To keep the pipeline programs generalized and reusable for different physical models, the interaction code is not implemented directly in the CH program. Instead, an approach using direct callables (DC) was chosen. They are linked to the pipeline through the SBT and act like normal functions where each of them is identifiable by an index.

To use the direct callables, a mapping from particle names to particle types has to be provided, where a type is just an unsigned integer. A second provided map then links particle types to direct callables, which dictate how a particle type interacts with the surface. These callables can then be called from within the pipeline.

An example would be a process that involves the particles A B and C. A and B both behave like neutral particles and are assigned to type 0, while C is ionized and belongs to type 1. The second mapping then links all particles of type 0 to one set of functions and particles of type 1 to a different set of functions that are fitting to their physical properties. Each particle type can have three different callables assigned to it, which describe the interaction at different stages of the ray's life:

- **Init callable:** Called when the ray is initialized in the RG program. Can be used to set particle specific parameters in `PerRayData`, as for example energy, and is optional.
- **Collision callable:** Called when the ray hits a surface element. This is where the contribution of the ray to the elements flux counter is calculated.
- **Reflection callable:** Called directly after the Collision callable. The ray weight reduction and reflection angle calculation are placed here.

Inside the pipeline programs, the current particle type is looked up and the related callable index in the SBT is determined, which is then used to call the correct direct callable. With this setup, the pipelines stay generalized, while having an option to provide all kinds of custom surface interactions externally.

Similar to the unnecessary fields in `PerRayData` described in Section 5.4, using direct callables also comes with a small performance overhead of about 1-2% compared to inlining code directly into the pipeline programs. This loss is outweighed by the gained flexibility and modularity, as different interactions can be implemented and tested without having to modify the main pipeline code.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

### 5.6.4 Warp Divergence and Shader Execution Reordering

GPUs rely on Single Instruction Multiple Thread (SIMT) execution, where groups of threads, called warps, execute the same instruction simultaneously. On NVIDIA hardware this warp size is typically 32. If threads in a warp take different control paths, the warp serializes the execution of each path, leading to severe performance degradation known as warp divergence. Ray tracing is inherently prone to warp divergence as the origins and directions of rays in a warp are randomly sampled, which can lead to multiple cases in which threads diverge. Typical causes are:

- Rays hitting boundary elements while others do not.
- Some rays in a warp hitting material A while others hit material B, where different surface interaction code has to be executed.
- Some rays reaching termination criteria earlier than others.

One way to reduce warp divergence is to assign each warp a set of rays with similar origins and directions. That way the rays are likely to take the same path, thus leading to more coherent execution. Manually sorting rays could introduce significant overhead and complexity, especially when dealing with millions of rays, which is why this method was not used here.

What was implemented instead is the use Shader Execution Reordering (SER), which was introduced with the Ada Lovelace GPU architecture (NVIDIA RTX 40 series) and OptiX 8.0. SER works by first running the ray-element-intersection tests and then using the available information about the hit primitive to rearrange threads, so they will take the same execution path in the upcoming CH shader [43]. The current implementation orders rays based on if they hit a boundary as well as if they are still active, meaning that their weight is still above the given threshold. The effect of this is very noticeable, when only simulating portions of the entire simulation domain and using symmetry in combination with boundary conditions to reduce computation time. In these cases, many rays will hit the boundary and being able to group them together reduces divergence significantly. Sorting out inactive rays is also beneficial as threads that hold a ray that was already terminated will not execute any further instructions, unless all rays in its warp come to and end. By grouping all inactive rays, a warp can finalize its ray generation loop sooner, and the scheduler can assign a new set of rays to the warp.

## 5.7 Post-Processing and Advection

After flux calculations are finished, the fluxes have to be normalized using the element's and the source plane's surface area in 3D or surface lengths in 2D. The area for a disk primitive can be trivially computed as the area of a circle with the disk radius. Elements that lie on the boundaries of the geometry can partially stick out of the simulation domain. In those cases the `DiskBoundingBoxXYIntersector`, implemented in ViennaRay, finds the portion of the disk that is outside the bounding box. Subtracting that from the full disk area gives the area inside the domain. For the line primitives, the length is simply the distance between the two endpoints, and for the triangles, the surface area can be computed using the cross product of the edge vectors.

For the area calculation of the surface inside a level-set voxel, there are multiple options. The method that was chosen in this implementation uses the SDF values at the corners and builds a polygon that approximates the surface inside the voxel [27]. This algorithm is very closely related to the marching cubes algorithm which is already used when extracting the line/triangle mesh. Although the goal of the level-set flux engine is to avoid explicit meshing, this method is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

relatively easy to implement and should give reasonable results in most cases. The algorithm starts by looking at the SDF values at each edge. If the sign changes along an edge, linear interpolation is used to find the zero-crossing point on the edge. Connecting all zero-crossing points yields a polygon, the area of which can be computed using the shoelace formula in 3D or simply by the distance between the points in 2D. This method assumes the surface to be linear and can be inaccurate if the surface is curved, as demonstrated in Fig. 5.12. Another issue is that ViennaLS uses Manhattan distances for the SDF values internally, which is not accounted for here.

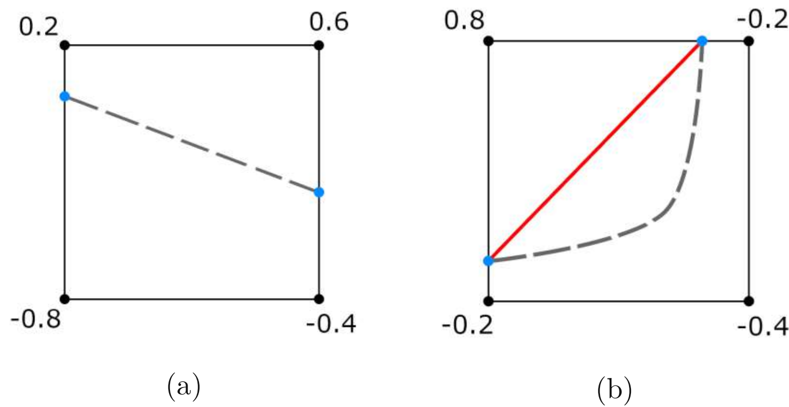



**Fig. 5.12:** Surface length calculation in a 2D voxel. The blue dots mark the zero-crossing points that are found by linearly interpolating along the edges. The gray line indicates the implicit surface. (a) The surface is approximated correctly. (b) The linear interpolation underestimates the surface length.

There are more advanced methods to calculate the surface area such as Dual Contouring [54]. They are more complex to implement and were not considered for this work. Another option would be to subdivide the voxel into many sub-voxels and find the surface area in each of those using the marching cubes algorithm before summing them up. With enough subdivisions, this should give accurate results even for curved surfaces. This method was also not implemented for this work, but could be an interesting option for the future.

After normalizing the fluxes, they are averaged over all first order neighbor elements to reduce noise and smooth the results. When doing this, the contributions of each neighbor are weighted by their difference of the normal vectors, where an identical normal gives full weight and a perpendicular one gives zero weight. For the triangle and disk mesh, the normals are directly available as they are stored per element during mesh extraction. The line and level-set mesh use the same methods as previously described in Section 5.5.2 and Section 5.5.3. For the line mesh, the normals are found by swapping the elements of the direction vector, while for the level-set mesh, the normals are computed by central differences in combination with linear interpolation. As a reference point for the central differences, the midpoint of each voxel is used.

The final fluxes can then be used to calculate advection velocities, depending on surface material and the used physical model. The element velocities are translated back to the level-set grid. The disk version uses a stored translator from the mesh extraction to find the corresponding active grid point for each disk. The line and triangle versions use a kdTree to find the closest grid point for each element. For the level-set mesh we iterate over the initial grid points, add the velocities of all voxels of which this point is a part, and then average them.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

## 5.8 Limitations and Potential Improvements

The current implementation focuses solely on accelerating the flux calculation using a GPU as it was by far the most time-consuming task. Other parts of the simulation pipeline are still executed on the CPU. This includes the meshing of the geometry, the level-set solver for updating the surface, the deposition/etch rate calculations, and coverage updates based on the fluxes obtained from the ray tracing step.

Meshing for the disk mesh is already very fast, porting it to the GPU is not a priority. The line/triangle mesh generation with its marching cubes algorithm is more time-consuming but still not the bottleneck of the overall simulation. There are existing algorithms for GPU-based marching cubes that could be used here, but real performance improvements are not guaranteed [55].

Solving the level-set equation could potentially benefit from a GPU implementation; however, the problem is that the current ViennaHRLE data structure is complex and not designed for use on a GPU. The entire ViennaHRLE structure would need to be adapted and ported to a GPU-friendly version which could be a considerable effort and is the topic for future work.

The rate and coverage calculations could be ported easily as they are done per surface element and do not have complex dependencies, but again the performance gain might not be significant enough to justify the effort, similar reasoning applies for post-processing. Generally speaking, the flux calculation is still the most demanding part and could further be optimized by:

- Reducing the number of rays by either finding an optimal number as a tradeoff between accuracy and performance, or by implementing adaptive ray counts based on local surface characteristics like curvature.
- Implementing a sorting system during ray generation to bundle rays with similar origins and directions together to reduce warp divergence.
- Ensuring coalesced memory access patterns for both ray and geometry data. This could include reorganizing geometry data to be spatially local in both memory and simulation space.

Regarding the level-set based implementation for GPU ray tracing, priority should be to improve the quality of the final surfaces as they are currently not on par with the disk and line/triangle mesh results, which will be discussed in the next chapter. The most likely reason for the poor results are errors in the post-processing step during area and normal vector calculations. More advanced methods, like Dual Contouring [54], or an adaptive refinement of the voxel grid near highly curved areas could be implemented.

Furthermore, the current level-set voxel mesh generator is a major bottleneck with very inefficient iteration over the ViennaHRLE structure. Due to the overall poor quality of final surfaces when using the level-set mesh covered in the next chapter, no further effort was made to optimize this.

An optimization that could benefit both the disk and level-set engines is to decrease the size of the boundary elements to perfectly fit around the simulation domain. Currently, both methods use elements that are larger than necessary to guarantee full coverage regardless of orientation of the simulated domain. Making the elements smaller could benefit BVH traversal times as less unnecessary intersection tests would be performed. The line and triangle engines already use perfectly fitting boundary elements. One possible approach would be to use mixed primitives, where the boundary elements are replaced with lines/triangles while keeping the disk/level-set elements for the actual surface.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

# 6 Results

In this chapter, the different flux engines will be benchmarked on a given set of example processes. The benchmarks will cover both 2D and 3D geometries with varying complexity to evaluate the performance and accuracy of each method. The final surfaces will be compared qualitatively by visual inspection and quantitatively by using ViennaLS to compare the surface area as well as the Chamfer distance between the active level-set grid points. The Chamfer distance is the sum of the distances from each point in one set to the nearest point in the other set and vice versa, providing a measure of similarity between two point clouds. At the time of writing ViennaLS only supports these quantitative metrics for two-dimensional domains. Most benchmarks were repeated 5 times and the median was taken, while for longer processes (>30s), the median of 3 runs was taken. For very long runs (>100s), two runs were performed, with the first one as a baseline and the second one to confirm the first result. Benchmark metrics will be the total process runtime as well as a relative speedup in total runtime compared to the CPU. The first benchmark also compares the performance in terms of rays per second. All benchmarks were executed on the same system with the specifications given in Tab. 6.1.

In general, the GPU flux engines should outperform the CPU disk engine in all benchmarks. The 2D GPU disk and line engines should be very similar in terms of runtime as the number of mathematic operations for intersection tests and the number of elements to represent a surface are very similar for both methods. In 3D, the triangle engine should be the fastest as OptiX leverages the available RT cores for ray-triangle intersection tests, which are significantly faster than using regular CUDA cores for the custom primitive intersection tests. The level-set engine is expected to be the slowest GPU engine in all cases due to the higher complexity of intersection tests as well more expensive post-processing.

| Component | Specification |
|---|---|
| OS | Ubuntu 24.04 |
| CPU | Intel Core i7-9700K |
| GPU | NVIDIA RTX 4070 |
| Intel Embree | v. 4.3.3 |
| CUDA | v. 12.4 |
| CUDA Driver | v. 550.144.03 |
| NVIDIA OptiX | v. 8.0 |

**Table 6.1:** System specifications used for benchmarking.

## 6.1 Comparison of CPU and GPU Disk Method

As the CPU and GPU disk methods use the same surface representation, the results should be equivalent up to differences introduced by the RNG used for sampling the ray origin and direction as well as the different neighbor detection methods. To verify this, the flux calculation was performed on a 2D trench geometry consisting of 239 tangential disks, shown in Fig. 6.1,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

and the number of hits at every disk was analyzed without any normalization or post-process neighbor averaging; the results are given in Fig. 6.2.

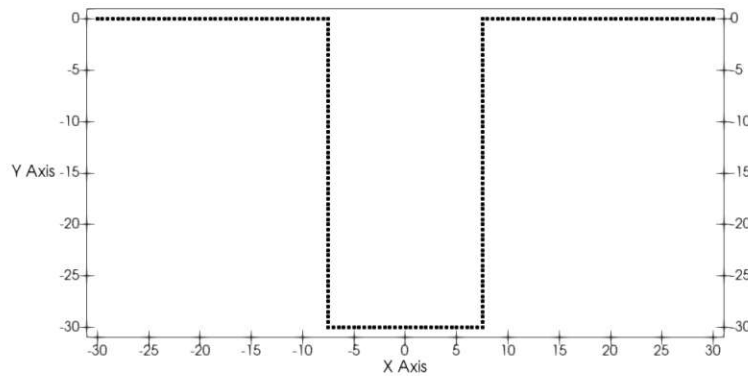



**Fig. 6.1:** 2D trench geometry used for CPU and GPU disk method comparison. Each dot is the center of a disk element.

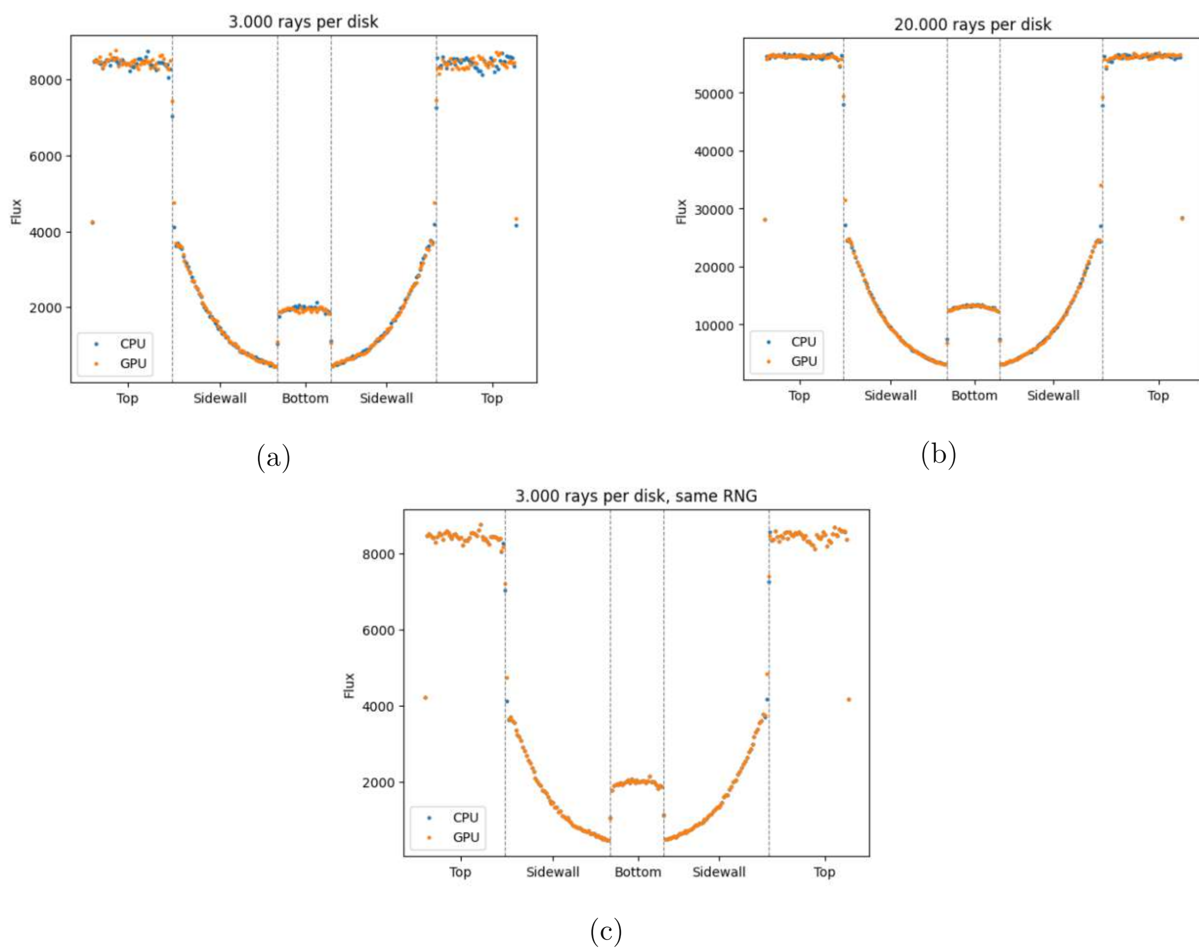



**Fig. 6.2:** Flux per disk in a 2D trench comparing CPU and GPU disk method with: (a) different RNG, 3,000 rays per disk, (b) different RNG, 20,000 rays per disk and (c) same RNG, 3,000 rays per disk.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As expected, Fig. 6.2a shows that the probabilistic nature of the sampling introduces some noise to the result. Note that the neighbor averaging, which was omitted here to gain a direct comparison, significantly reduces this noise in actual simulations. By increasing the number of rays, the noise decreases and the results become more similar as seen in Fig. 6.2b. By fixing the seed for the RNG and writing the configuration of each ray used in the CPU version to a file, then reading that file on the GPU version, both methods can be compared using the exact same rays. When doing this, the results match perfectly everywhere other than at the corners because of the different neighbor detection, as seen in Fig. 6.2c. The fluxes at the corners between top and sidewall are slightly higher in the GPU version due to less shadowing as depicted in Fig. 5.7.

In theory, this should lead to a slightly higher total flux on the GPU version as more rays can reach the sidewalls; however, this should not have a significant impact on the overall results. The neighbor averaging in the post-processing step further decreases these differences. In the following benchmarks, the CPU and GPU disk methods are expected to produce nearly identical results.

## 6.2 Single Particle Deposition

This benchmark compares the raw ray tracing performance in terms of rays per second with minimal influence from process specific variation. The results can be seen as a maximum achievable improvement in runtime for the given geometry, while still inside the ViennaPS framework. Different geometries might achieve different results.

In this example, the geometry is a cylindrical hole in 3D and a trench in 2D. We simulate a simple particle deposition process with one particle type. The particle behaves like a neutral particle with diffuse reflections and a constant sticking probability of 0.1. This value was chosen to cause many reflections, which increases the load on the flux engines. The surface does not have any physical material properties that could affect the particle behavior.

The initial geometry, final geometry from the CPU disk method, and exact parameters are given in Fig. 6.3 and Tab. 6.2. The number of active level-set grid points in the initial geometry is 601 in 2D and 47,003 in 3D, which directly translates to the number of disks. This number is used to determine the total number of rays per iteration regardless of the actual number of surface elements to ensure a fair comparison between the different flux engine types. In 3D, the amount of active grid points is multiplied by a factor of 3,000 which has shown to create results with acceptable low noise. In 2D, this factor is doubled to 6,000 as the ray tracing in 2D is already very efficient. This results in a total of 3.6 million rays per iteration in 2D and 140 million rays per iteration in 3D. The amount of initial and final surface elements for each mesh type is given in Tab. 6.3. The process is executed for 10 iterations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

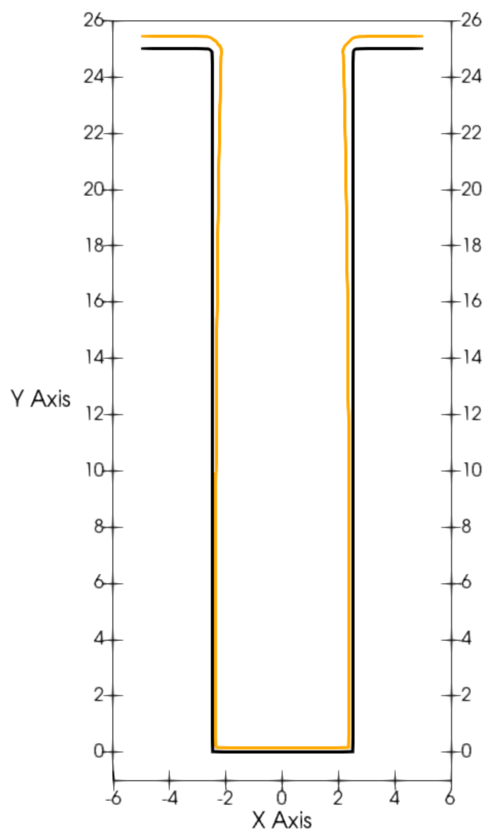



**Fig. 6.3:** 2D single particle deposition initial and final surface (CPU disk); black: initial trench, orange: deposited layer on top.

| Parameter | Value |
|---|---:|
| gridDelta | 0.1 |
| xExtent | 10 |
| yExtent | 10 |
| trenchWidth | 5 |
| trenchHeight | 25 |
| taperAngle | 0 |
| rate | 1 |
| stickingProbability | 0.1 |
| sourcePower | 1 |
| particleType | Neutral |
| processSteps | 10 |
| integrationScheme | E.-O. 1st order |

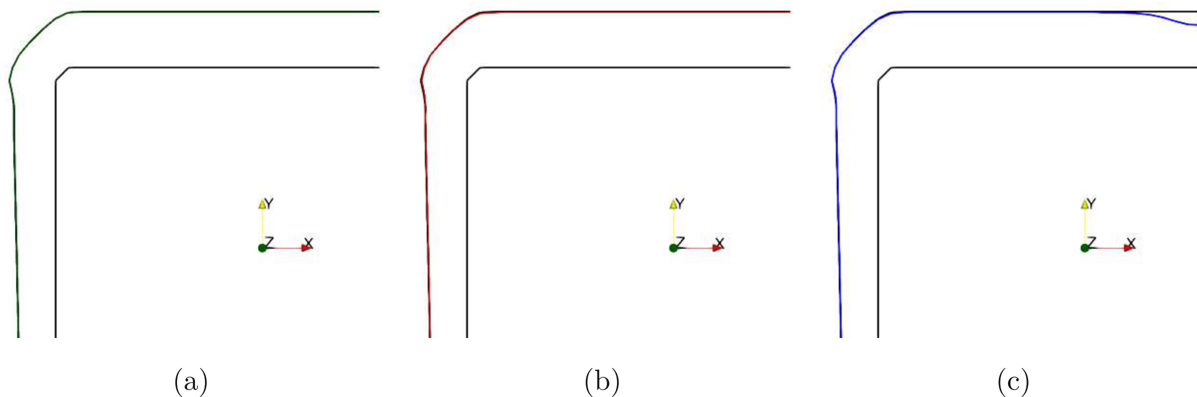
**Table 6.2:** Single particle deposition benchmark configuration.

| | 2D | | 3D | |
|---|---:|---:|---:|---:|
| **Method** | Initial | Final | Initial | Final |
| CPU Disks | 601 | 603 | 47,003 | 44,758 |
| GPU Disks | 601 | 603 | 47,003 | 44,761 |
| Level-set voxels | 600 | 615 | 58,892 | 59,911 |
| Lines | 600 | 614 | - | - |
| Triangles | - | - | 109,962 | 117,503 |

**Table 6.3:** Single particle deposition initial and final surface element count.

Visually all resulting surfaces are near identical in both 2D and 3D to the reference CPU disk solution. Because of this, only the 2D results are shown in Fig. 6.4. The only noticeable difference is that the level-set method produces a downwards bend towards the +X in 2D and both the +X and +Y boundaries in 3D. No specific cause for this behavior has been identified yet and has to be investigated in the future.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) (b) (c)

**Fig. 6.4:** 2D single particle deposition final surfaces (top right corner), black: initial geometry and CPU disk, (a) GPU disk, (b) lines, (c) level-set.

The surface difference metrics in Tab. 6.4 are both very low for the GPU disk method, which indicates that the results are also close in a quantitative manner. However, the values for the line method, especially the area difference, are noticeably higher and do not reflect the visual similarity of the surfaces. The metrics for the level-set method are omitted here, as the results were orders of magnitude larger than from the other methods. This could be due to the bending which causes large local differences and seemingly very inaccurate metric calculations. The following examples will not have this kind of problem.

| **Method** | Area Difference | Chamfer Distance |
|---|---|---|
| CPU Disks | - | - |
| GPU Disks | 0.09 | 0.0031 |
| Level-set | - | - |
| Lines | 5.81 | 0.0772 |

**Table 6.4:** 2D Surface difference metrics.

The runtimes given in Tab. 6.5 show significant speedups for all GPU methods compared to the CPU reference. As expected, the 2D disk and line methods perform very similar with a speedup of 24x-25x. The level-set method, which is computationally more expensive, only achieves a speedup of 5.5x compared to the CPU in 2D. In 3D, the triangles are the fastest with a speedup of 54x. As OptiX is optimized for triangle meshes it is not surprising that they outperform the other methods. The disk method also achieved a significant improvement in runtime of around 34x while the level-set method is again the slowest, but still a lot faster than the CPU with a factor of approximately 10x.

Very importantly, this benchmark was performed with a fixed number of rays per iteration. In subsequent benchmarks, the number of rays will be relative to the number of surface elements, where a triangle mesh usually has around twice as many elements as the other meshes. Also, the process specific overhead was kept minimal here, which will not be the case in later examples. The more complex physical models require more work on the CPU side, which will reduce the relative impact of the ray tracing performance. Therefore, it is expected that the differences between the methods will be less pronounced in later benchmarks

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

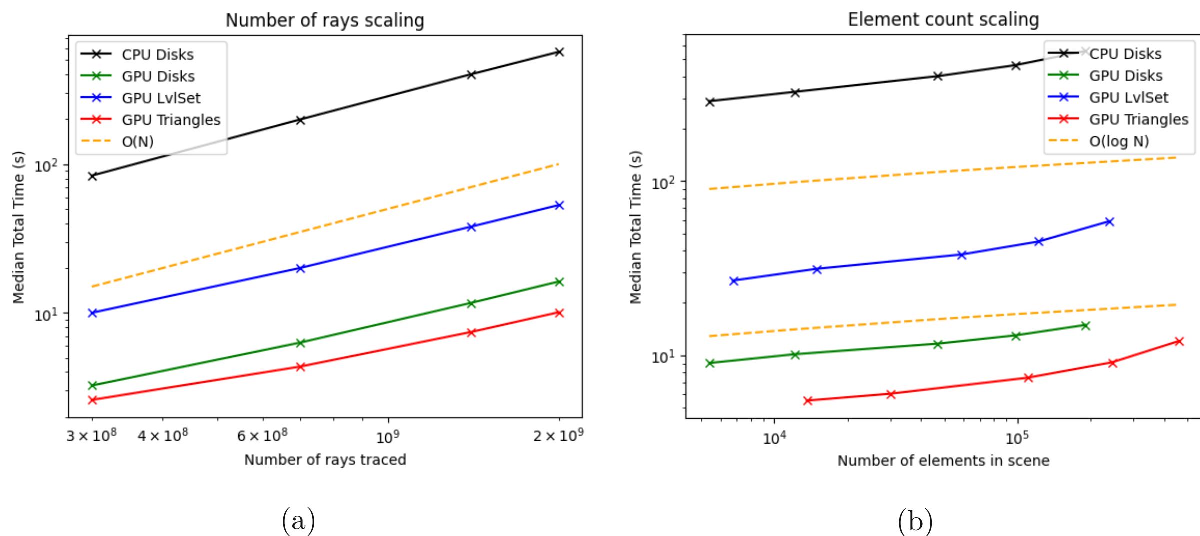
| | 2D 36M rays | | | 3D 1.4B rays | | |
|---|---|---|---|---|---|---|
| **Method** | Time | M Rays/s | Speedup | Time | M Rays/s | Speedup |
| CPU Disks | 7.887s | 4.565 | - | 401.08s | 3.491 | - |
| GPU Disks | 0.308s | 116.88 | 25.61x | 11.670s | 119.97 | 34.37x |
| Level-set | 1.423s | 25.29 | 5.54x | 37.947s | 36.89 | 10.57x |
| Lines | 0.327s | 110.09 | 24.12x | - | - | - |
| Triangles | - | - | - | 7.437s | 188.25 | 53.93x |

**Table 6.5:** Single particle deposition benchmark.

### 6.2.1 Scalability with Number of Surface Elements and Rays

When keeping the grid delta fixed and varying the number of rays as shown in Fig. 6.5a, only the load on the ray tracing engines changes, so the runtime scales linearly with the number of rays, as one would expect.

Fig. 6.5b shows the scaling with the number of surface elements, which was achieved by varying the grid delta. The element count has a direct impact on the ray tracing engines as BVH traversal gets more demanding, but other parts of the processing pipeline, such as meshing and advection, are affected as well. In the best case, BVH traversal scales logarithmically with the number of elements and linearly in the worst case. The tested scene seems to be close to logarithmic scaling as the runtime only increases slightly when increasing the number of elements. Even with other parts of the pipeline being affected, the overall scaling is still close to logarithmic and definitely sublinear.




(a) (b)

**Fig. 6.5:** 3D single particle deposition scalability with (a) number of rays and (b) number of surface elements.

## 6.3 Hole Etching - $SF_6/O_2$ Plasma Etching

The hole etching example in ViennaPS is a more complex process that simulates $SF_6/O_2$ plasma etching, which is different from the previous benchmark in that it uses multiple particle types as well as more sophisticated particle surface interactions. The surface material is now taken into account, where silicon can be removed while the mask material is not affected by the process. In

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

total, three particle types are used: ions, etchant, and oxygen, where the last two behave like neutral particles. The surface interactions are implemented as described in Section 4.4. The geometry is again a cylindrical hole in 3D and a trench in 2D, but only half of the geometry is simulated using the symmetry in combination with reflective boundary conditions to reduce runtime. The initial geometry, final geometry from the CPU disk method and exact parameters are given in Fig. 6.6 and Tab. 6.6. As mentioned, the number of rays will now be relative to the number of surface elements which can be seen in Tab. 6.7.

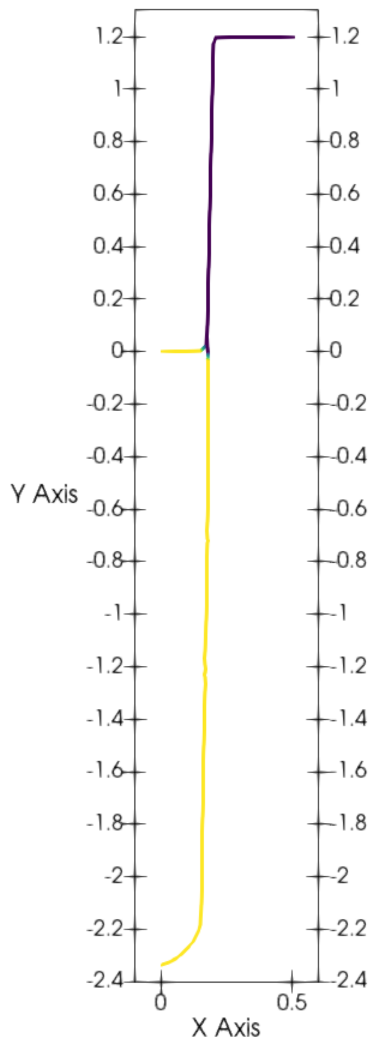



**Fig. 6.6:** Hole etching initial and final surface (CPU disk); purple: mask, yellow: silicon.

| Parameter | Value |
|---|---|
| lengthUnit | um |
| gridDelta | 0.03 |
| xExtent | 1.0 |
| yExtent | 1.0 |
| holeRadius | 0.175 |
| maskHeight | 1.2 |
| taperAngle | 1.193 |
| processTime | 1 |
| timeUnit | min |
| ionFlux | 10 |
| etchantFlux | 4.5e3 |
| oxygenFlux | 8e2 |
| ionExponent | 1000 |
| meanEnergy | 100 eV |
| sigmaEnergy | 10 eV |
| $A_O$ | 2 |
| $A_{Si}$ | 7 |
| etchStopDepth | -10 |
| integrationScheme | E.-O. 1st order |
| raysPerPoint | 1000 |

**Table 6.6:** Hole etching benchmark configuration.

| | 2D | | 3D | |
|---|---|---|---|---|
| **Method** | Initial | Final | Initial | Final |
| CPU Disks | 57 | 131 | 1,353 | 2,502 |
| GPU Disks | 57 | 131 | 1,353 | 2,496 |
| Level-set voxels | 57 | 143 | 1,568 | 2,747 |
| Lines | 57 | 146 | - | - |
| Triangles | - | - | 3,078 | 6,710 |

**Table 6.7:** Initial and final element count.

Fig. 6.7 shows that the line and level-set methods etch slightly deeper than the disk methods in 2D. This is likely due to the disk mesh having elements on the corners that can shadow incoming rays, preventing them from reaching further down the trench (s. Fig. 5.4). Lowering the grid delta and making the disk elements smaller reduces this effect. This was done for the GPU disk method in Fig. 6.7b, resulting in a final etch depth much closer to the other methods. The same

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

effect can be observed in 3D as shown in Fig. 6.8. This could indicate that the disk methods are less accurate in the case of high aspect ratio features, given the same grid delta. On the other hand, the roughness of the final surface from both the line and level-set methods is more significant compared to the disk methods. Also, the level-set method produces a very different profile with a much shallower etch depth and a slightly wider shape in 3D.

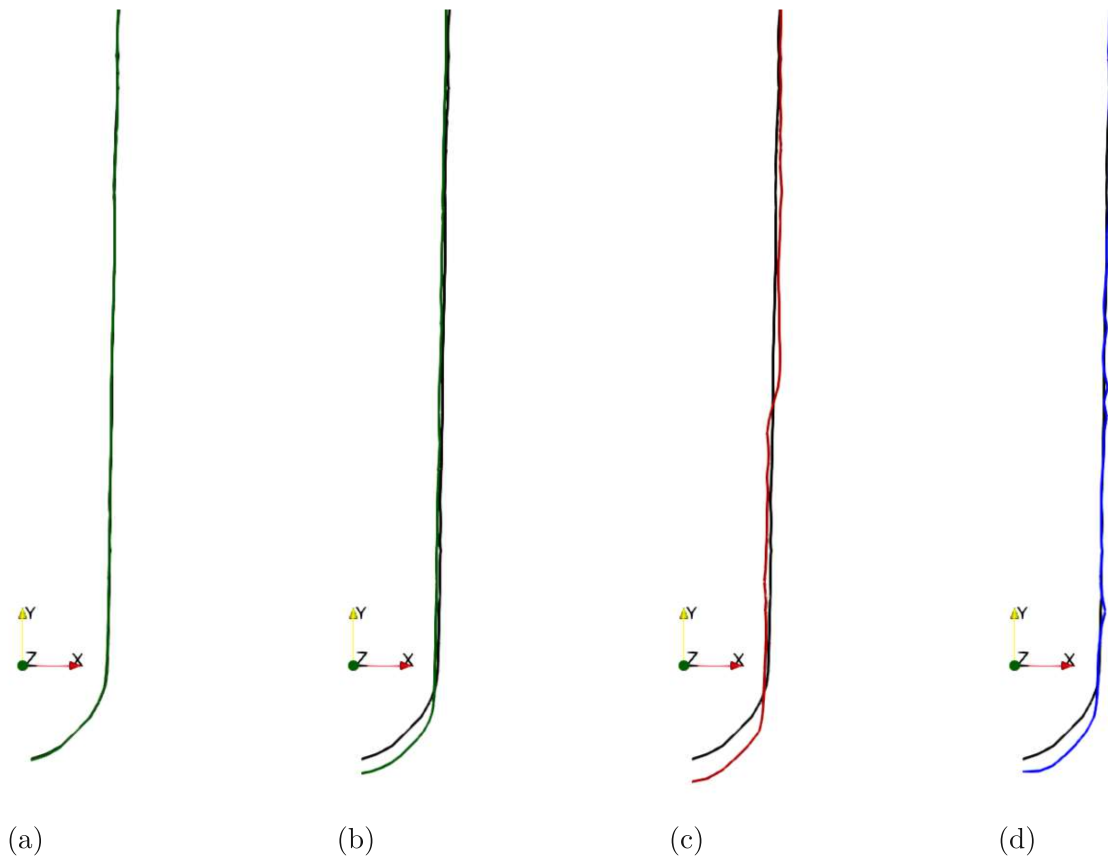



**Fig. 6.7:** 2D hole etching final surfaces, black: CPU disk, (a) GPU disk, (b) GPU disk; gridDelta=0.02, (c) lines, (d) level-set.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

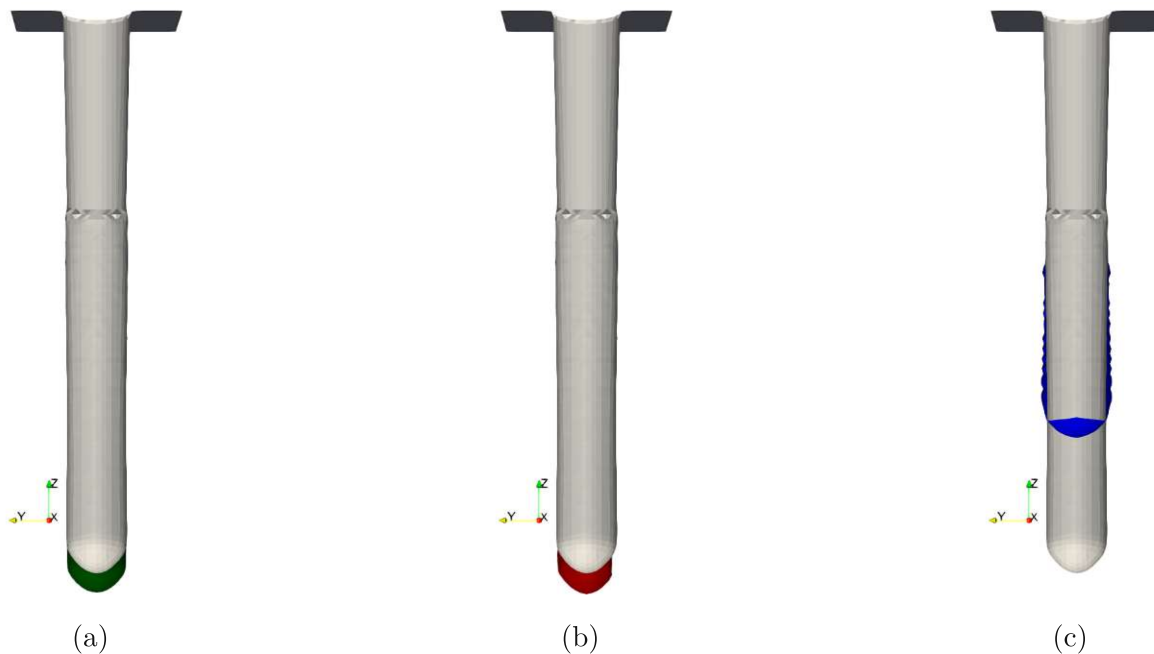


Fig. 6.8: 3D hole etching final surfaces, white: CPU disk, (a) GPU disk; gridDelta=0.02, (b) triangles, (c) level-set.

The surface metrics given in Tab. 6.8 confirm that the GPU disk method produces practically identical results to the CPU reference. The different etch depths of the other methods is also reflected in the metrics, where both the line and level-set methods have slightly higher values for the area difference and Chamfer distance.

| **Method** | Area Difference | Chamfer Distance |
|---|---|---|
| CPU Disks | - | - |
| GPU Disks | 0 | 0.0008 |
| Level-set | 0.0126 | 0.0039 |
| Lines | 0.0117 | 0.0057 |

**Table 6.8:** 2D Surface difference metrics.

The runtimes given in Tab. 6.9 again behave similarly for the disks and lines with both achieving a speedup of almost 9x in 2D, while the level-set is, as in the previous benchmark, the slowest. A currently unsolved problem in the implementation for the level-set flux engine is the very slow conversion from the level-set grid to the voxel mesh, which is then used for ray tracing. Reading the material IDs poses a bottleneck due to poor use of iterators, which should be further investigated. Because of the poor final surface quality in this and the upcoming benchmarks, solving this was not prioritized in this work.

In 3D, the GPU disk method was faster than the triangle method even though the previous benchmark showed that for a triangle mesh more rays per time interval can be traced. The reason for this is that the triangle engine traces around 2.5x more rays than the disk engine, since the simulation was set to have a specific number of rays per element and there are 2.5x more triangles than disks, as seen in Tab. 6.7. Therefore, the final runtime is longer despite being able to trace rays faster. Future work should determine if a triangle mesh with the same

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

number of total rays would yield final surfaces with similar accuracy and therefore be a better choice overall.

| | 2D | | 3D | |
|---|---|---|---|---|
| **Method** | Time | Speedup | Time | Speedup |
| CPU Disks | 12.096s | - | 179.83s | - |
| GPU Disks | 1.356s | 8.92x | 8.480s | 21.21x |
| Level-set | 4.974s | 2.43x | *35.904s | 5.01x |
| Lines | 1.399s | 8.65x | - | - |
| Triangles | - | - | 12.083s | 14.88x |

*Level-set final surface in 3D is significantly different.

**Table 6.9:** Hole etching benchmark.

## 6.4 Bosch Process - Deep Reactive Ion Etching

The Bosch process cycles through etching and passivation steps to create deep trenches with vertical sidewalls. In total 10 cycles are simulated. This benchmark will have a significantly higher number of surface elements compared to the previous hole etching example as can be seen in Tab. 6.11. The initial geometry, final geometry from the CPU disk method and exact parameters are again given in Fig. 6.9 and Tab. 6.10.

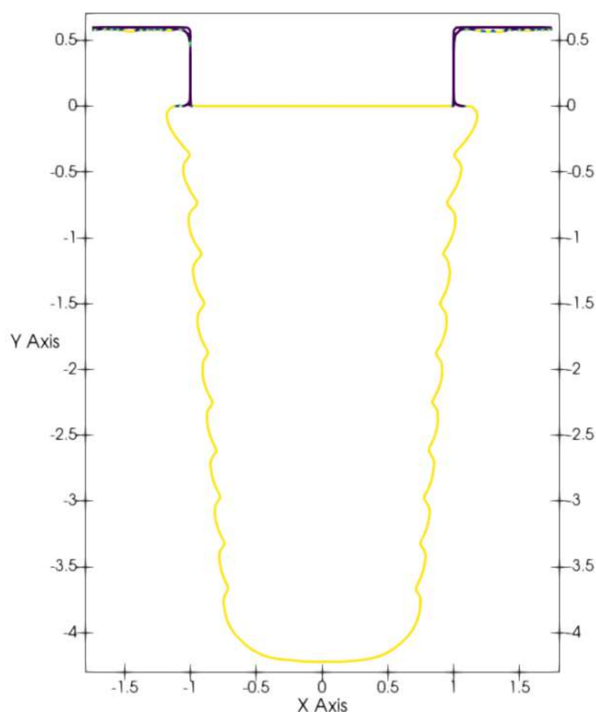



**Fig. 6.9:** Bosch process initial and final surface (CPU disk); purple: mask, yellow: silicon.

| **Parameter** | **Value** |
|---|---|
| lengthUnit | um |
| gridDelta | 0.025 |
| xExtent | 3.5 |
| yExtent | 3.5 |
| trenchWidth | 2.0 |
| depositionStickingProb. | 0.01 |
| depositionThickness | 0.075 |
| neutralStickingProb. | 0.1 |
| neutralRate | -0.2 |
| ionSourceExponent | 200 |
| ionRate | -0.1 |
| etchTime | 1.5 |
| numCycles | 10 |
| integrationScheme | E.-O. 1st order |
| raysPerPoint | 1000 |

**Table 6.10:** Bosch process benchmark configuration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

| | 2D | | 3D | |
|---|---|---|---|---|
| **Method** | Initial | Final | Initial | Final |
| CPU Disks | 189 | 485 | 26,649 | 68,383 |
| GPU Disks | 189 | 485 | 26,649 | 68,385 |
| Level-set | 188 | 638 | 26,320 | 88,110 |
| Lines | 188 | 642 | - | - |
| Triangles | - | - | 52,080 | 179,995 |

**Table 6.11:** Initial and final element count.

Since the CPU and GPU disk methods have a different shadowing behavior at corners, as described in Section 5.5.1.1, the GPU method has a slightly higher total flux on the sidewalls leading to a bit more horizontal etching, similar to the results of the line engine, shown in Fig. 6.10. The level-set engine can barely represent the characteristic scalloped sidewalls of the Bosch process and also produces a wider final trench. The surface metrics in Tab. 6.12 show that the GPU disk and line methods are not exactly the same because of the deeper scallops, compared to the CPU reference while both are in the same order of magnitude. The level-set method, on the other hand has a significantly higher area difference and Chamfer distance. Fig. 6.11 shows the final surface of the level-set method in 3D, compared to the CPU disk reference, which is of very poor quality and does not resemble the expected trench shape. A bend on the +Y boundary can once again be observed, similar to the previous benchmark result from Fig. 6.4. The 3D disk and triangle comparisons are omitted here as they show the same characteristics as in 2D with no further insights.

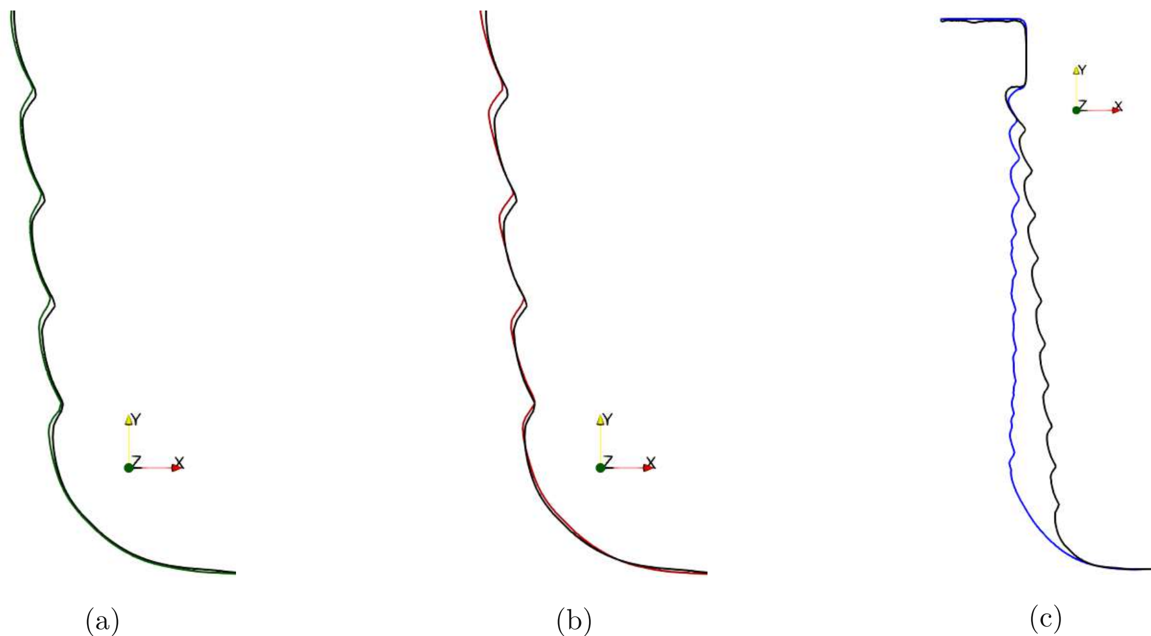



**Fig. 6.10:** 2D Bosch process final surfaces, black: CPU disk, (a) GPU disk, (b) lines, (c) level-set.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

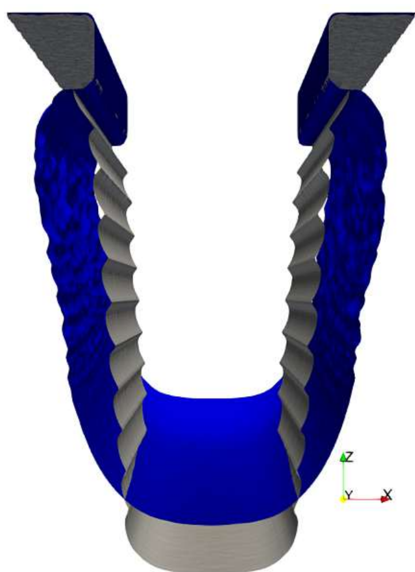



**Fig. 6.11:** 3D final surface, white: CPU disk, blue: level-set.

| Method | Area Difference | Chamfer Distance |
|---|---|---|
| CPU Disks | - | - |
| GPU Disks | 0.0369 | 0.0042 |
| Level-set | 1.6525 | 0.1317 |
| Lines | 0.0938 | 0.0096 |

**Table 6.12:** 2D Surface difference metrics.

In this example, the pattern for the runtimes, presented in Tab. 6.13, stays the same with disks and lines performing similarly in 2D while the level-set method is around 2.5x slower. In 3D, the disk engine is again faster than the triangles because of fewer elements and rays. The large number of surface elements in 3D causes a high load on the flux engines, which is why the speedups are higher than in the previous benchmark and closer to the results found in Tab. 6.5.

This benchmark clearly shows the advantage of using a GPU based ray tracing engine as the simulation time is reduced from almost 2.5 hours to under 6 minutes with the disk method, while producing nearly identical or, regarding the shadowing effects, arguably better results than the CPU.

| | **2D** | | **3D** | |
|---|---|---|---|---|
| **Method** | Time | Speedup | Time | Speedup |
| CPU Disks | 36.792s | - | 146min | - |
| GPU Disks | 3.895s | 9.45x | 328.76s | 26.66x |
| Level-set | 10.649s | 3.45x | 974.79s | 8.99x |
| Lines | 4.454s | 8.26x | - | - |
| Triangles | - | - | 503.03s | 17.42x |

**Table 6.13:** Bosch process benchmark.

## 6.5 DRAM Wiggling - $HBr/O_2$ Plasma Etching

The DRAM Wiggling example in ViennaPS simulates $HBr/O_2$ plasma etching on an even larger simulation domain compared to the previous benchmarks and is only available in 3D. The number of elements, which are given in Tab. 6.15, are more than doubled compared to the Bosch process benchmark, which makes this example computationally very intensive. The initial geometry and used parameters are given in Fig. 6.12 and Tab. 6.14.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

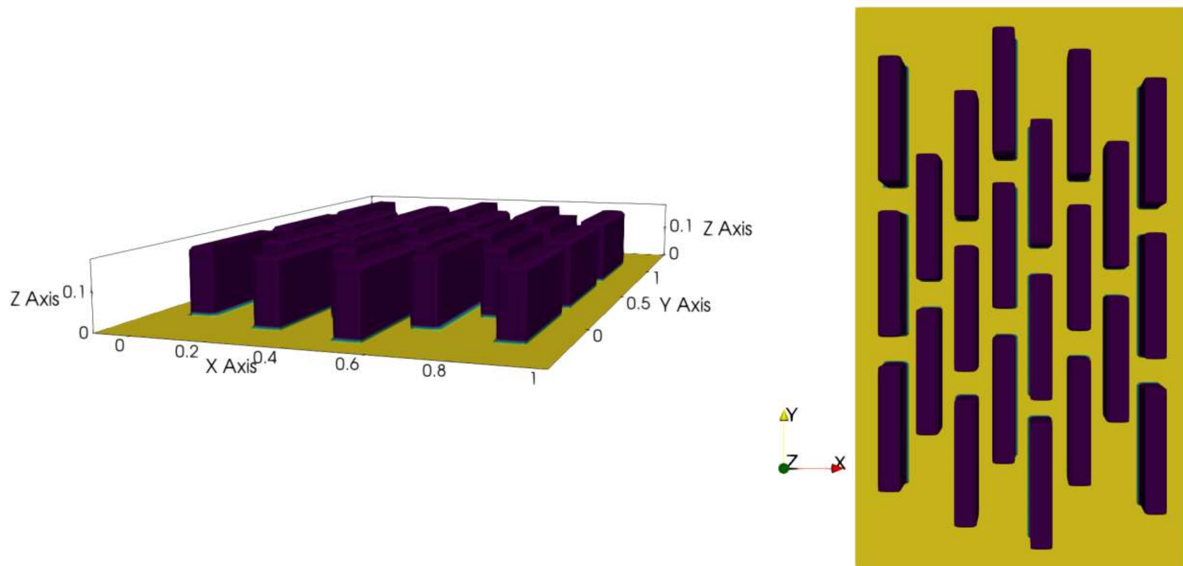



**Fig. 6.12:** 3D DRAM Wiggling initial surface; purple: mask, yellow: silicon.

| Parameter | Value |
|---|---:|
| lengthUnit | um |
| gridDelta | 0.01 |
| xExtent | 1.11 |
| yExtent | 1.86 |
| timeUnit | second |
| processTime | 20 |
| ionFlux | 10 |
| etchantFlux | 4.5e3 |
| oxygenFlux | 2.5e3 |
| ionExponent | 1000 |
| meanEnergy | 200 eV |
| sigmaEnergy | 10 eV |
| integrationScheme | Lax-F. 2nd order |
| raysPerPoint | 1000 |

**Table 6.14:** DRAM Wiggling benchmark configuration.

| | 3D | |
|---|---:|---:|
| **Method** | Initial | Final |
| CPU Disks | 57,139 | 155,455 |
| GPU Disks | 57,139 | 155,191 |
| Level-set | 57,788 | 192,358 |
| Triangles | 110,108 | 381,985 |

**Table 6.15:** Initial and final element count.

Opposing to the previous examples, the surface produced with the level-set engine matches the other solutions very well, as seen in Fig. 6.13. One reason for this could be that the flat surfaces and sharp 90 degree angles can be represented well by the voxel grid, while also leading to accurate area and normal calculations; however, there is again an upwards bend on the +X and +Y boundaries. The GPU disk and triangle engines produce very similar results, while the final surface from the triangle engine shows slightly less material removal between the fins. The results from the CPU disk engine were again practically identical to the GPU version and are omitted here.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

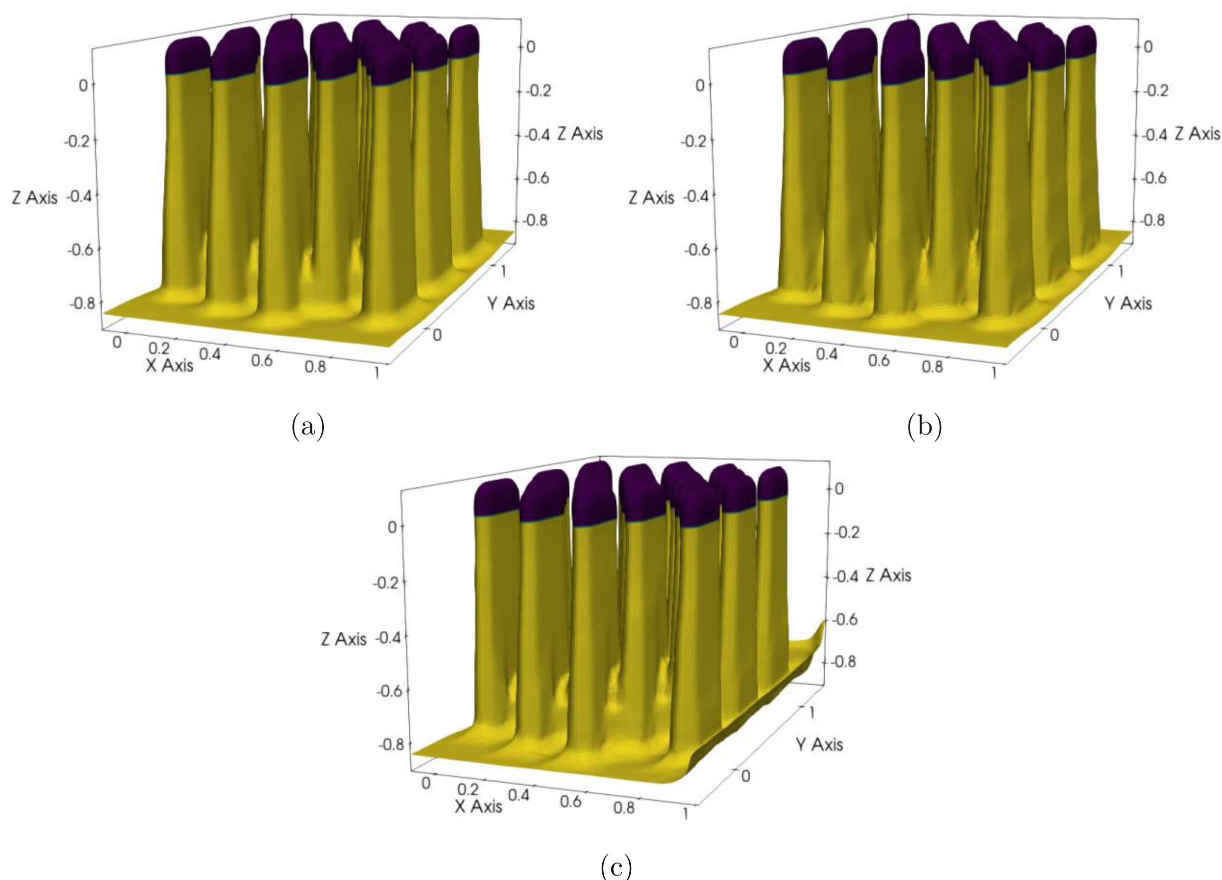



**Fig. 6.13:** 3D DRAM Wiggling final surfaces, purple: mask, yellow: silicon; (a) GPU disk, (b) triangles, (c) level-set.

When taking a slice of the surfaces at the XY-plane at Z = -0.5, and comparing to the shape of the fins at the start of the simulation, as shown in Fig. 6.14, a wiggling effect is clearly visible. The fins are bulging outwards in areas where more etchant is required, as it was described in Section 3.2.3. While the shapes produced by the disk and triangle engines are nearly indistinguishable, the level-set engine shows slightly less pronounced wiggling and overall a less consistent shape across neighboring fins.

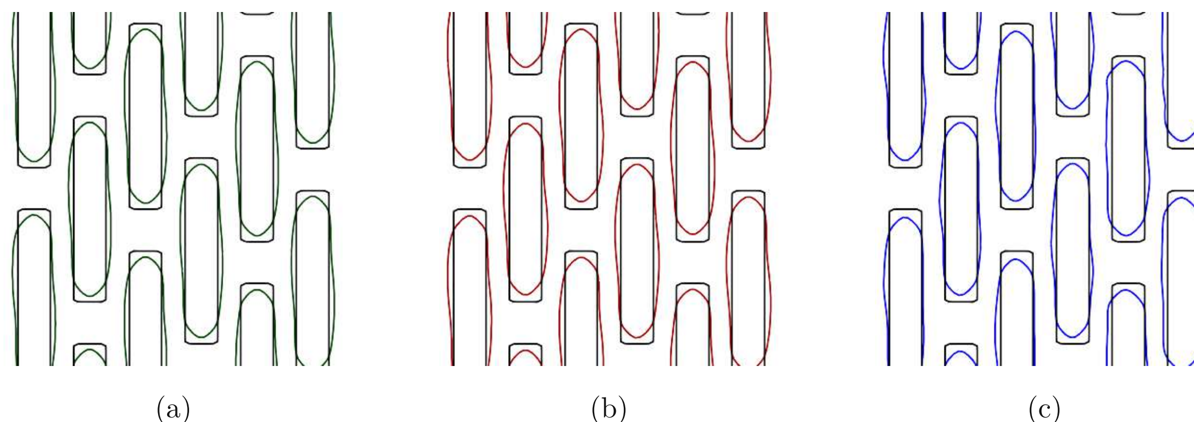



**Fig. 6.14:** XY-plane at Z=-0.5, black: initial fin shape; (a) GPU disk, (b) triangles, (c) level-set.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Compared to the previous Bosch process benchmark, the number of initial and final surface elements is more than doubled here, which further increases the relative performance advantage of the GPU based flux engines, as seen in Tab. 6.16. The GPU disk method is 34x faster than the CPU version, while producing practically identical results. This shows that for such large and complex simulations, the GPU based flux engines are practically a necessity to obtain results in reasonable time.

| | **3D** | |
|---|---|---|
| **Method** | Time | Speedup |
| CPU Disks | 10.7h | - |
| GPU Disks | 19min | 34.00x |
| Level-set | 167min | 3.86x |
| Triangles | 33min | 19.49x |

**Table 6.16:** DRAM Wiggling benchmark.

## 6.6 Profiling

In order to better understand the performance differences between the four surface representation methods, the ray tracing pipelines were profiled using NVIDIA Nsight Compute v. 2024.1.1. The profiling focuses on two main areas: Streaming Multiprocessor (SM) utilization and memory performance. The SM profiling metrics provide insights into how effectively the GPU's computational resources are being used, while the memory profiling metrics hint at potential bottlenecks related to memory access patterns and bandwidth utilization. While the areas are separated here for clarity, they are inherently interconnected, as inefficient use of either can impact overall performance. The values presented in Tab. 6.17 and Tab. 6.18 are the average over all ray tracing kernel launches during the examples runtime. In the case of simulations with multiple particles, this average can deviate from the actual measured values, as neutral and charged particles behave differently. Internally, the separate kernels from the ray tracing pipeline (RG, IS, CH, etc.) all get compiled into a single kernel, so analysis of specific sections is not trivial. RT core utilization metrics are not included in this analysis, as they are not available through Nsight Compute.

The following metrics with a short explanation and their abbreviation were collected for each benchmark and surface reconstruction method:

- **SM Metrics:**
  - `sm__throughput.avg.pct_of_peak_sustained_elapsed` (ThrP. [%]):
    Measures the percentage of the theoretical maximum throughput achieved by the SMs. A low value indicates that the computational resources are not fully utilized, which could be due to various factors such as memory bottlenecks.
  - `smsp__thread_inst_executed_per_inst_executed.ratio` (T./warp):
    Indicates the average number of active threads in each warp during execution and is therefore a measurement for warp divergence. Optimally this value should be 32, meaning all threads in a warp are active, taking the same control path and contributing to the computation. The worst case is 1 in which each thread takes a separate branch. In this case every execution has to be serialized which drastically reduces performance.
  - `tpc__average_registers_per_thread.ratio` (Reg./T.):
    Shows the average number of registers allocated per thread. A high register usage can limit the number of active warps per SM, reducing occupancy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

  - sm__warps_active.avg.pct_of_peak_sustained_active (Occ. [%]): Represents the ratio of active warps to the maximum number of warps supported on an SM (occupancy). A low occupancy does not necessarily mean poor performance, but a higher occupancy means that the scheduler can better hide memory latency by switching between warps when some are waiting for memory operations to complete. The RTX 4070 used for the benchmarks supports a maximum of 48 active warps and 65.536 registers per SM. In the case of 128 registers per thread, the occupancy is limited by the number of registers as only $\frac{65536}{128 \times 32} = 16$ warps can be active per SM. This results in a theoretical maximum occupancy of $\frac{16}{48}$ which is 33%.

- **Memory Metrics:**
  - l1tex__t_sector_hit_rate.pct (L1 Hit [%]): Measures the percentage of memory accesses that hit in the L1 cache, indicating how effectively the cache is being utilized.
  - lts__t_sector_hit_rate.pct (L2 Hit [%]): Measures the percentage of memory accesses that are served by the L2 cache.
  - dram__bytes.sum.per_second (DRAM [GB/s]): Amount of data transferred between the L2 cache and the GPUs main memory per second. A high value means that a lot of data is loaded from main memory, which is significantly slower than L2 or L1 cache access.
  - l1tex__average_t_sectors_per_request_pipe_lsu_mem_global_op_ld.ratio (Unc R.): Ratio of transferred bytes to requested bytes for global memory read operations. Ideally this value should be close to 1, while the worst case is 32. A higher number indicates inefficient, uncoalesced memory access patterns that can lead to performance degradation.
  - l1tex__average_t_sectors_per_request_pipe_lsu_mem_global_op_st.ratio (Unc W.): The same for write operations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

| | Example | Method | ThrP. [%] | T./warp | Reg./T. | Occ. [%] |
|---|---|---|---|---|---|---|
| 2D | Deposition | Disks | 29.14 | 22.99 | 113.75 | 32.73 |
| | | Level-set | 13.97 | 19.73 | 128.87 | 33.20 |
| | | Lines | 27.42 | 23.99 | 114.05 | 32.70 |
| | Hole etching | Disks | 24.58 | 12.02 | 117.41 | 26.68 |
| | | Level-set | 18.03 | 12.07 | 126.65 | 31.06 |
| | | Lines | 25.17 | 13.48 | 115.02 | 27.20 |
| | Bosch | Disks | 25.58 | 20.90 | 114.22 | 32.03 |
| | | Level-set | 13.27 | 18.63 | 128.50 | 33.05 |
| | | Lines | 25.74 | 22.27 | 114.47 | 32.26 |
| | | | | | | |
| 3D | Deposition | Disks | 31.18 | 14.25 | 113.13 | 33.04 |
| | | Level-set | 15.12 | 13.74 | 129.16 | 33.22 |
| | | Triangles | 21.47 | 29.85 | 114.01 | 32.77 |
| | Hole etching | Disks | 31.25 | 13.82 | 113.59 | 32.23 |
| | | Level-set | 17.88 | 14.56 | 129.22 | 33.01 |
| | | Triangles | 25.81 | 26.73 | 114.30 | 32.43 |
| | Bosch | Disks | 29.06 | 17.04 | 113.22 | 33.00 |
| | | Level-set | 13.57 | 16.53 | 129.30 | 33.22 |
| | | Triangles | 22.50 | 29.59 | 113.90 | 32.80 |
| | DRAM | Disks | 28.77 | 15.74 | 113.17 | 33.01 |
| | | Level-set | 14.94 | 14.44 | 129.43 | 33.20 |
| | | Triangles | 22.42 | 27.71 | 113.96 | 32.78 |

**Table 6.17:** SM profiling metrics for all methods.

Tab. 6.17 shows a recurring pattern across all examples in both 2D and 3D. The generally low SM Throughput is primarily due to the offloading of BVH traversal to the RT Cores, which are not included in the SM metrics. The SMs have to wait for the RT cores to report potential intersections with AABBs, leading to low utilization.

The number of active threads per warp has already been drastically improved by Shader Execution Reordering as mentioned in Section 5.6.4. For example in the 3D variant of the hole etching benchmark, SER lifted the number of active threads per warp from originally $\approx 5$ to now $\approx 14$ for the disk and level-set methods, resulting in a more than twice as fast runtime. The triangle mesh engine shows even less warp divergence, as the intersection calculations are also executed on the RT cores, where they cannot cause any warp divergence. Given the fact that the triangle engine is already very coherent and close to the optimum of 32, further improvements through SER are limited, as it can only reduce divergence caused by the CH program. Since the CH program is mostly identical for all methods, the focus for reducing warp divergence further should be on the intersection programs of the different surface representation methods.

All methods require a relatively high number of registers per thread, and each kernel is executed with 512 threads per block, a value chosen by the compiler. This limits the amount of active warps per SM to 16, which is one third of the possible maximum. This maximum is reached in almost every case, except for the 2D hole etching example with the disk and line methods, where no concrete explanation could be found.

The level-set method uses more than 128 registers/thread on average in all but one examples, which could indicate register spilling; however, the number could also come from accumulation of errors from averaging over the available data. If register spilling does occur, it could be another

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

reason for the lower performance of the level-set method, as spilled registers are stored in local memory, which is much slower than register access. This could also explain the lower throughput achieved in every example.

| | Example | Method | L1 Hit [%] | L2 Hit [%] | DRAM [GB/s] | Unc R. | Unc W. |
|---|---|---|---|---|---|---|---|
| 2D | Deposition | Disks | 41.94 | 99.91 | 116.22 | 5.96 | 23.98 |
| | | Level-set | 22.37 | 79.98 | 360.22 | 6.96 | 29.97 |
| | | Lines | 34.16 | 99.68 | 139.38 | 7.20 | 23.99 |
| | Hole etching | Disks | 67.89 | 101.97 | 21.52 | 2.86 | 22.65 |
| | | Level-set | 37.60 | 87.52 | 227.52 | 3.89 | 28.50 |
| | | Lines | 63.66 | 99.26 | 31.42 | 3.02 | 23.11 |
| | Bosch | Disks | 44.61 | 99.68 | 87.35 | 5.66 | 23.11 |
| | | Level-set | 19.65 | 79.34 | 341.44 | 7.63 | 29.05 |
| | | Lines | 39.63 | 99.69 | 109.02 | 6.19 | 23.42 |
| | | | | | | | |
| 3D | Deposition | Disks | 39.36 | 99.93 | 77.74 | 5.79 | 24.01 |
| | | Level-set | 28.16 | 81.33 | 349.80 | 7.51 | 29.65 |
| | | Triangles | 21.85 | 98.89 | 162.12 | 15.63 | 23.99 |
| | Hole etching | Disks | 51.02 | 100.04 | 70.95 | 4.31 | 22.73 |
| | | Level-set | 23.11 | 81.65 | 351.43 | 5.88 | 28.18 |
| | | Triangles | 27.81 | 99.61 | 145.34 | 8.95 | 22.81 |
| | Bosch | Disks | 37.82 | 99.92 | 81.29 | 6.40 | 23.57 |
| | | Level-set | 21.18 | 80.74 | 347.01 | 8.69 | 29.67 |
| | | Triangles | 24.84 | 98.95 | 152.58 | 14.75 | 24.18 |
| | DRAM | Disks | 36.67 | 99.86 | 83.65 | 6.10 | 23.20 |
| | | Level-set | 23.97 | 80.66 | 348.70 | 7.48 | 27.89 |
| | | Triangles | 22.88 | 98.60 | 158.86 | 14.61 | 22.82 |

**Table 6.18:** Memory profiling metrics for all methods.

Tab. 6.18 shows a consistent pattern across all examples in both 2D and 3D. A probable cause for the lower performance across all benchmarks of the level-set method is its poor cache utilization, as both L1 and L2 cache hit rates are lower than those of the other methods. This leads to more frequent access to the slower DRAM, which is also reflected in the higher DRAM bandwidth usage. A plausible explanation is that the level-set method requires more data per element, filling up the L2 cache faster. As memory is limited on the L2 cache, fewer elements can be stored there. This reduces the probability of a cache hit and instead the data has to be fetched from the much slower main memory.

The metrics indicating uncoalesced memory accesses are also worse for the level-set engine. Either way, the memory access is not optimal for any of the methods, presenting an area for potential future optimizations. Possible improvements include restructuring data layouts to ensure 16-byte alignment as this could improve cache line utilization. In the current implementation, many values are stored in arrays of structs, where each struct represents a 3D vector with 3 separate float values, leading to 12 byte alignment. Adding a padding float to reach 16 byte alignment could in theory improve cache line usage. Ensuring coalesced memory accesses when millions of random rays are traced is inherently difficult, but another potential improvement could be to sort rays based on their origin and direction before assigning them to a warp. If the data layout of the geometry is then also spatially organized, this could increase the likelihood

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

that threads within a warp access nearby memory locations, further improving cache utilization and reducing memory latency.

## 6.7 Energy Efficiency

One additional benchmark metric presented here is the energy efficiency of the CPU and GPU flux engines. For that, the DRAM Wiggling example from before is rerun for a simulated process time of 2 seconds and the average power consumption and utilization is measured; the results are shown in Tab. 6.19. The example was chosen as it is very ray tracing heavy, so both the CPU and GPU can work on ray tracing with as few interruptions as possible. The metrics on the CPU were measured by using the `turbostat` tool on Linux which uses the CPUs RAPL (Running average power limit) interface that provides power consumption data. The GPUs metrics were measured with `nvidia-smi`. NVIDIA claims that the power measurements from `nvidia-smi` are accurate within 5 W. The utilization is the percentage of time the device was actively working on the program, it is not an indicator on how efficiently the device is using its resources.

The comparison will be between the CPU and GPU disk methods as they produce practically the same results. The power consumption for the other GPU methods should be similar as the ray tracing load is comparable.

| **Device** | Time | Avg. CPU Power | Avg. GPU Power | Energy |
|---|---|---|---|---|
| CPU | 1049.58s | 95.55 W | - | 27.86 Wh |
| CPU + GPU | 31.05s | 35.23 W | 108.78 ± 5 W | 1.24 ± 0.04 Wh |

**Table 6.19:** Average power draw and energy consumption of CPU and GPU disk flux engines for the DRAM Wiggling example (processTime: 2s).

Using only the CPU, the average utilization was at 93.3%, whereas when using CPU and GPU combined, the average CPU utilization dropped to 18.5% as the GPU took over the ray tracing with a utilization of 81.2%.

The CPU ran in line with its TDP of 95W while the GPU was significantly below its TDP of 200W which is not surprising as the simulations are mostly memory-bound and the Streaming Multiprocessors do not fully utilize their compute capabilities. Also, separate sections of the GPU, such as the encoder and decoder or the tensor cores are not used at all.

While the total combined power draw when doing the ray tracing on the GPU is around 1.5x higher compared to just the CPU, the total energy consumption is more than 22x lower due to the significantly reduced runtime. This shows that using the GPU for ray tracing is not only faster, but also several times more energy efficient. Given the fact that GPUs are generally more expensive to purchase, the cost of acquisition could potentially be amortized over time due to the lower energy consumption.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

# 7 Conclusion and Future Directions

The disk based flux engine, which runs on the CPU in ViennaPS, was successfully ported to the GPU, achieving significant speedups in both 2D (9x-25x) and 3D (21x-34x) simulations. The GPU implementation was validated against the existing CPU version, showing that the GPU version uses a better method of detecting overlapping disk hits that leads to more continuous fluxes on corners and confirming that it produces identical results elsewhere, while also greatly reducing computation time.

The already existing triangle based flux engine for 3D simulations was also extended with a 2D counterpart that represents surfaces as lines. Both the triangle and line method show a slightly higher calculated flux due to less shadowing effects on corners, compared to the disk method. The shadowing effects were also confirmed in the disk version by lowering the size of the disks and observing final results that are more in line with the triangle and line engines.

The line engine achieved similar performance (8x-24x) as the disk engine in 2D, which was expected, given the similar number of surface elements needed to represent geometries as well as similar computational complexity of intersection tests. In 3D, the triangle engine is able to trace 1.5x more rays per second than the disk engine due to hardware acceleration through dedicated ray tracing cores present on modern NVIDIA GPUs. This advantage is compensated by the need for approximately 2.5x more surface elements to represent geometries, as well as the number of rays scaling with the number of elements. Because of this, the disk engine came out faster in overall runtime despite the lower raw ray tracing performance. It is unclear if the comparison using rays per surface element is fair or if a fixed number of rays should be used instead. Doing so would likely favor the triangle method, but it has to be tested if the accuracy and noise remain acceptable when lowering the number of rays.

An attempt was made to avoid explicit meshing by using the provided signed distance values of the level-set grid provided by ViennaLS directly, but the results were very inconsistent. In 2D, the simulation results are somewhat reasonable in some examples, while in 3D the generated surfaces are of very poor quality with significant artifacts and deviations from the expected results. One reoccurring issue was an upwards bending of the surface at the +X and +Y boundaries which could not be resolved.

A possible explanation for the poor results could be the area calculation for normalization, which can approximate the surface area incorrectly on curved surfaces. Supporting this thesis are the surprisingly good results on a geometry with flat surfaces and sharp angles, as given in the last benchmark example. Due to producing poor results, no effort was placed into optimizing the level-set method, which lead to inferior performance compared to the other GPU engines, but still a speedup of 2.4x-5.5x in 2D and 4x-10x in 3D, compared to the CPU reference. Possible approaches to improve the quality of the level-set results, such as better area and normal calculations through methods like Dual Contouring or adaptive grid refinement were discussed.

The GPU methods were profiled using Nsight Compute, showing potential memory boundedness caused by uncoalesced memory accesses and in the case of the level-set engine, potential register spilling. Possible optimizations to ensure more coherent memory access were discussed, such as ensuring memory alignment of data structures by padding, or sorting rays based on their origin and direction to increase the likelihood of accessing similar regions in memory.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

If the ViennaPS user has access to an NVIDIA GPU, using the GPU flux engines is highly recommended due to the significant speedups achieved in all benchmarks. GPUs from other vendors are currently not supported as ViennaPS relies on NVIDIA OptiX. Future work could include porting the implementation to a more vendor-agnostic framework like Vulkan RT or OpenCL to support a wider range of hardware, though OpenCL lacks support for use of dedicated ray tracing hardware present on modern GPUs. The GPU disk engine can be used as a drop-in replacement for the existing CPU disk engine, while the line and triangle engines can achieve similar performance with slightly different results. Either way, the engines need to be calibrated for the specific use case to ensure accurate results.

The performance improvements open up new possibilities for simulating more complex processes or geometries within reasonable timeframes. Simulations that previously took several hours can now be completed in a few minutes, enabling faster iteration and optimization of fabrication processes. This is especially beneficial during the development phase of new processes where many simulations with slightly varied parameters are required to find optimal settings.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Bibliography


[1] Institute for Microelectronics TU Wien. *Software Information*. 2018. URL: https://www.iue.tuwien.ac.at/viennacl0 (visited on 12/23/2025).

[2] T. Reiter and L. Filipovic. "ViennaPS: A flexible framework for semiconductor process simulation". In: *SoftwareX* 32 (2025), p. 102453. DOI: 10.1016/j.softx.2025.102453.

[3] T. Reiter and L. Filipovic. "Fast 3D Flux Calculation using Monte Carlo Ray Tracing on GPUs". In: *Proceedings of the International Conference on Microelectronic Devices and Technologies (MicDAT '2023)*. Ed. by S. Y. Yurish. 2023, pp. 67–72. DOI: 10.13140/RG.2.2.13265.71524.

[4] D. Antoniadis and R. Dutton. "Models for computer simulation of complete IC fabrication process". In: *IEEE Transactions on Electron Devices* 26.4 (1979), pp. 490–500. DOI: 10.1109/T-ED.1979.19452.

[5] M. Stettler, A. Slepko, S. Smith, V. Tiwari, C. Weber, J. Weber, S. Cea, S. Hasan, L. Jiang, A. Kaushik, P. Keys, R. Kotlyar, C. Landon, and D. Pantuso. "State-of-the-art TCAD: 25 years ago and today". In: *2019 IEEE International Electron Devices Meeting (IEDM)*. 2019, pp. 39.1.1–39.1.4. DOI: 10.1109/IEDM19573.2019.8993451.

[6] Synopsys. *Synopsys TCAD*. 2025. URL: https://www.synopsys.com/manufacturing/tcad.html (visited on 12/23/2025).

[7] Silvaco. *Semiconductor Process and Device Simulation*. 2025. URL: https://www.silvaco.com/tcad/ (visited on 12/23/2025).

[8] Global TCAD Solutions. *GTS Framework*. 2025. URL: https://www.globaltcad.com/products/gts-framework/ (visited on 12/23/2025).

[9] Lam Research. *SEMulator3D*. 2025. URL: https://www.lamresearch.com/product/semulator3d/ (visited on 12/23/2025).

[10] R. J. Hoekstra and M. J. Kushner. *Monte Carlo Feature Profile Model (MCFPM)*. 2025. URL: https://cpseg.eecs.umich.edu/Projects/MCFPM/MCFPM.htm (visited on 12/23/2025).

[11] S. Yao. "SimProfile: A Monte Carlo Surface Profile Simulator with Data-Driven Parameter Calibration". In: *Preprints* (2025). DOI: 10.20944/preprints202510.1834.v1.

[12] Synopsys. *Quantum ATK*. 2025. URL: https://www.synopsys.com/manufacturing/quantumatk.html (visited on 12/23/2025).

[13] W. Van Roosbroeck. "Theory of the flow of electrons and holes in germanium and other semiconductors". In: *The Bell System Technical Journal* 29.4 (1950), pp. 560–607. DOI: 10.1002/j.1538-7305.1950.tb03653.x.

[14] A. Martinez, N. Seoane, M. Aldegunde, A. Asenov, and J. R. Barker. "The Non-equilibrium Green function approach as a TCAD tool for future CMOS technology". In: *2011 International Conference on Simulation of Semiconductor Processes and Devices*. 2011, pp. 95–98. DOI: 10.1109/SISPAD.2011.6035058.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[15] M. Duane. “The Role of TCAD in Compact Modeling”. In: *TechConnect Briefs: Technical Proceedings of the 2002 International Conference on Modeling and Simulation of Microsystems*. Vol. 1. TechConnect, 2002, pp. 719–721.

[16] K. Rupp. *Microprocessor Trend Data*. 2022. URL: https://github.com/karlrupp/microprocessor-trend-data/tree/master (visited on 12/23/2025).

[17] NVIDIA. *GeForce RTX 5090*. 2025. URL: https://www.nvidia.com/en-us/geforce/graphics-cards/50-series/rtx-5090/ (visited on 12/23/2025).

[18] M. Wagner, K. Rupp, and J. Weinbub. “A comparison of algebraic multigrid preconditioners using graphics processing units and multi-core central processing units”. In: *Proceedings of the 2012 Symposium on High Performance Computing*. HPC ’12. Orlando, Florida: Society for Computer Simulation International, 2012, pp. 99–106.

[19] LAMMPS. *GPU package - LAMMPS Documentation*. 2024. URL: https://docs.lammps.org/Speed_gpu.html (visited on 12/23/2025).

[20] OpenACC. *Quantum Espresso On GPUs*. 2025. URL: https://www.openacc.org/sites/default/files/inline-images/Presentations/Quantum%20Espresso%20on%20GPUs%20-%20OpenACC%20Webinar.pdf (visited on 12/23/2025).

[21] M. Dighamber. “Physics-Informed Deep Learning for Plasma Etch Optimization”. MA thesis. Cambridge: Massachusetts Institute of Technology, 2024.

[22] J. Bobinac. “Process Simulation and Model Development in ViennaPS”. MA thesis. Vienna: TU Wien, 2023.

[23] L. Filipovic. “Topography Simulation of Novel Processing Techniques”. Dissertation. Vienna: TU Wien, 2012.

[24] L. Sun, G. Yuan, L. Gao, J. Yang, M. Chhowalla, M. Heydari Gharahcheshmeh, K. Gleason, Y. Choi, B. Hong, and Z. Liu. “Chemical vapour deposition”. In: *Nature Reviews Methods Primers* 1 (2021), p. 5. DOI: 10.1038/s43586-020-00005-y.

[25] R. Prabu, S. Ramesh, M. Savith, and M. Balachandar. “Review of Physical Vapour Deposition (PVD) Techniques”. In: *International Conference of Sustainable Manufacturing*. 2013, pp. 427–434. DOI: 10.13140/RG.2.1.5063.4964.

[26] R. Johnson, A. Hultqvist, and S. Bent. “A brief review of atomic layer deposition: From fundamentals to applications”. In: *Materials Today* 17 (2014), pp. 236–246. DOI: 10.1016/j.mattod.2014.04.026.

[27] C. Heitzinger. “Simulation and Inverse Modeling of Semiconductor Manufacturing Processes”. Dissertation. Vienna: TU Wien, 2002.

[28] R. W. Fiordalice, R. I. Hegde, and H. Kawasaki. “Orientation Control of Chemical Vapor Deposition TiN Film for Barrier Applications”. In: *Journal of The Electrochemical Society* 143.6 (1996), pp. 2059–2063. DOI: 10.1149/1.1836949.

[29] Institute for Microelectronics TU Wien. *ViennaPS*. 2025. URL: https://github.com/ViennaTools/ViennaPS (visited on 12/23/2025).

[30] K. Karimi, A. Fardoost, and M. Javanmard. “Comprehensive Review of FinFET Technology: History, Structure, Challenges, Innovations, and Emerging Sensing Applications”. In: *Micromachines* 15 (2024), p. 1187. DOI: 10.3390/mi15101187.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[31] M. Said, A. Shalaby, F. Mehdipour, M. Biglari-Abhari, and M. El-Sayed. “A design methodology and various performance and fabrication metrics evaluation of 3D Network-on-Chip with multiplexed Through-Silicon Vias”. In: *Microprocessors and Microsystems* 43 (2016). Many-Core System-on-Chip Architectures and Applications (PDP 15), pp. 26–46. DOI: 10.1016/j.micpro.2016.01.011.

[32] G. Schropfer and M. McNie. “Designing manufacturable MEMS in CMOS-compatible processes: Methodology and case studies”. In: *MEMS, MOEMS, and Micromachining, Proceedings of the International Society for Optics and Photonics, Strasbourg, France, 26-30 April 2004*. Ed. by H. Urey and A. El-Fatatry. Vol. 5455. International Society for Optics and Photonics. SPIE, 2004, pp. 116–127. DOI: 10.1117/12.544971.

[33] Faraday Technology. *Design Flow on FinFET Platform*. 2025. URL: https://www.faraday-tech.com/en/content/Product/ASIC_Service/FinFETPlatform (visited on 12/23/2025).

[34] J. Bobinac, T. Reiter, J. Piso, X. Klemenschits, O. Baumgartner, Z. Stanojevic, G. Strof, M. Karner, and L. Filipovic. “Effect of Mask Geometry Variation on Plasma Etching Profiles”. In: *Micromachines* 14.3 (2023), p. 665. DOI: 10.3390/mi14030665.

[35] O. Ertl. “Numerical methods for topography simulation”. Dissertation. Vienna: TU Wien, 2010.

[36] M. Haass, M. Darnon, G. Cunge, and O. Joubert. “Silicon etching in a pulsed HBr/O2 plasma. II. Pattern transfer”. In: *Journal of Vacuum Science & Technology B Microelectronics and Nanometer Structures* 12.118 (2015), p. 032203. DOI: 10.1116/1.4917231.

[37] S. Gomez, R. Belen, M. Kiehlbauch, and E. Aydil. “Etching of high aspect ratio features in Si using SF6/O2/HBr and SF6/O2/Cl2 plasma”. In: *Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films* 23 (2005), pp. 1592–1597. DOI: 10.1116/1.2049303.

[38] D. Kim, Y. K. Kim, and H. Lee. “A study of the role of HBr and oxygen on the etch selectivity and the post-etch profile in a polysilicon/oxide etch using HBr/O2 based high density plasma for advanced DRAMs”. In: *Materials Science in Semiconductor Processing* 10.1 (2007), pp. 41–48. DOI: 10.1016/j.mssp.2006.08.027.

[39] Q. Wang, Y. De Chen, J. Huang, and E. Joseph. “A Study of Wiggling AA modeling and Its Impact on the Device Performance in Advanced DRAM”. In: *2020 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*. 2020, pp. 101–104. DOI: 10.23919/SISPAD49475.2020.9241640.

[40] X. Klemenschits, S. Selberherr, and L. Filipovic. “Modeling of Gate Stack Patterning for Advanced Technology Nodes: A Review”. In: *Micromachines* 9.12 (2018), p. 631. DOI: 10.3390/mi9120631.

[41] Institute for Microelectronics TU Wien. *ViennaPS*. 2025. URL: https://viennatools.github.io/ViennaPS (visited on 12/23/2025).

[42] NVIDIA. *NVIDIA Turing Architecture In-Depth*. Sept. 14, 2018. URL: https://developer.nvidia.com/blog/nvidia-turing-architecture-in-depth/ (visited on 12/23/2025).

[43] NVIDIA. *NVIDIA OptiX 8.1 - Programming Guide*. Version 1.17. Oct. 22, 2024. URL: https://raytracing-docs.nvidia.com/optix8/guide/optix_guide.241022.A4.pdf (visited on 12/23/2025).

[44] Intel. *Intel® Embree High Performance Ray Tracing Kernels*. 2025. URL: https://raw.githubusercontent.com/embree/embree/master/readme.pdf (visited on 12/23/2025).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[45] C. Ericson. “Chapter 6 - Bounding Volume Hierarchies”. In: *Real-Time Collision Detection*. Ed. by C. Ericson. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 235–284. DOI: 10.1016/B978-1-55860-732-3.50011-5.

[46] Y. Hu, W. Wang, D. Li, Q. Zeng, and Y. Hu. “Parallel BVH Construction Using Locally-Density Clustering”. In: *IEEE Access* 7 (2019), pp. 105827–105839. DOI: 10.1109/ACCESS.2019.2932151.

[47] NVIDIA. *Why are some of my any hits missed?* 2024. URL: https://forums.developer.nvidia.com/t/why-are-some-of-my-any-hits-missed/288077 (visited on 12/23/2025).

[48] NVIDIA. *How to Get Started with OptiX 7*. Nov. 20, 2019. URL: https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/ (visited on 12/23/2025).

[49] X. Klemenschits. “Emulation and simulation of microelectronic fabrication processes”. Dissertation. Vienna: TU Wien, 2022. DOI: 10.34726/hss.2022.89324.

[50] G. Marmitt, A. Kleer, I. Wald, H. Friedrich, and P. Slusallek. “Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing”. In: *Vision, Modeling, and Visualization 2004 (VMV-04)*. 2004, pp. 429–435.

[51] H. Hansson Söderlund, A. Evans, and T. Akenine-Möller. “Ray Tracing of Signed Distance Function Grids”. In: *Journal of Computer Graphics Techniques (JCGT)* 11.3 (2022), pp. 94–113.

[52] G. Chatzianastasiou and G. A. Constantinides. “An Efficient FPGA-based Axis-Aligned Box Tool for Embedded Computer Graphics”. In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 2018, pp. 343–344. DOI: 10.1109/FPL.2018.00065.

[53] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. “A Survey of Voxel Interpolation Methods and an Evaluation of Their Impact on Volumetric Map-Based Visual Odometry”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3637–3643. DOI: 10.1109/ICRA.2018.8461227.

[54] T. Ju, F. Losasso, S. Schaefer, and J. Warren. “Dual contouring of hermite data”. In: *ACM Transactions on Graphics* 21.3 (2002), pp. 339–346. DOI: 10.1145/566654.566586.

[55] R. Geiss. “Generating Complex Procedural Terrains Using the GPU”. In: *GPU Gems 3*. Ed. by H. Nguyen. Addison-Wesley Professional, 2007, pp. 7–37. URL: https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu (visited on 12/23/2025).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub