UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Ingeniería Electrónica

PhD Thesis

Study, Modelling and Implementation of the Level Set Method Used in Micromachining Processes

Author:

Carles Montoliu Alvaro

Thesis advisors:

Dr. Joaquín Cerdá Boluda

Dr. Ricardo Colom Palero

Valencia, September 2015.

“*Sí com lo taur se’n va fuyt pel desert*
quant és sobrat per son semblant qui·l força,
ne torna may fins ha cobrada força
per destruir aquell qui l’ha desert:
tot enaxí·m cové lunyar de vós,
car vostre gest mon esforç ha confús:
no tornaré fins del tot haja fus
la gran pahor qui·m toll ser delitós.”

Ausiàs March, ⋆1397 - †1459

Per a Marc

Agradecimientos

Sin duda, todo el trabajo que hay detrás de esta tesis ha sido posible gracias a la colaboración y dedicación de muchos. Todos ellos merecen ser agradecidos por haber contribuido a su manera en la culminación de este trabajo.

Agradecer sinceramente a mis directores y tutores, Ximo Cerdà y Ricardo Colom, por permitir formarme como investigador y ofrecerme sus consejos. Ellos me han tratado con respeto y nunca han puesto freno a mis ambiciones, más bien las han engrandecido. Gracias por guiar toda esta aventura.

Una mención muy especial se merecen Miguel Ángel Gosálvez y Néstor Ferrando. Les tengo que agradecer la gran suerte que he tenido de poder colaborar con dos eminencias como ellos. Sus grandes conocimientos han permitido obtener los mejores resultados de este trabajo, además de haber sido una experiencia gratificante. Gracias por su constante apoyo y su ayuda infinita.

Asimismo, dar las gracias también a Vicente Herrero, Jose María Monzó, Ana Ros y Ramón J. Aliaga. Ellos dieron vida a un ambiente de trabajo idóneo, aportándome mucho más que ayuda y conocimientos. Del mismo modo Ángel Tébar, Rafael Gadea y Jorge Daniel siempre han estado dispuestos a ayudar en todo cuanto han podido.

Mi estancia en el instituto Fraunhofer de Erlangen me ha permitido acelerar mi formación y vivir unas experiencias de lo más satisfactorias. Todo ello ha sido gracias a la magnífica gente con la que he tenido el placer de colaborar y convivir. Gracias a Eberhard Bär y a toda la incontable gente del Fraunhofer IISB.

Un mérito especial se merecen mi familia y mis amigos por aguantarme todo este tiempo sin perder la paciencia conmigo ni en los peores momentos. Es de un valor incalculable tener tantos hombros en los que poder dejarte caer para coger las fuerzas suficientes para seguir. Gracias Isidor y Lourdes, gracias Ferran y Andreu, y gracias Vicente, Enric y David.

No podía faltar mi particular *aimia*, Nuria. Ella se merece un eterno agradecimiento por haber tenido que vivir todo esto conmigo. Compañera de todas y cada una de las experiencias, buenas y malas, de esta larga aventura. Animándome en todo momento, siempre ayudándome con un altruismo deslumbrante y enseñándome una inmensidad de conocimientos. Ella, ha sido y es, mi *llir entre cards.*

Una vez más, muchísimas gracias a todos.

Summary

The main topic of the present thesis is the improvement of fabrication processes simulation by means of the Level Set (LS) method. The LS is a mathematical approach used for evolving fronts according to a motion defined by certain laws. The main advantage of this method is that the front is embedded inside a higher dimensional function such that updating this function instead of directly the front itself enables a trivial handling of complex situations like the splitting or coalescing of multiple fronts.

In particular, this document is focused on wet and dry etching processes, which are widely used in the micromachining process of Micro-Electro-Mechanical Systems (MEMS). A MEMS is a system formed by mechanical elements, sensors, actuators, and electronics. These devices have gained a lot of popularity in last decades and are employed in several industry fields such as automotive security, motion sensors, and smartphones.

Wet etching process consists in removing selectively substrate material (e.g. silicon or quartz) with a liquid solution in order to form a certain structure. This is a complex process since the result of a particular experiment depends on many factors, such as crystallographic structure of the material, etchant solution or its temperature. Similarly, dry etching processes are used for removing substrate material, however, gaseous substances are employed in the etching stage.

In both cases, the usage of a simulator capable of predicting accurately the result of a certain experiment would imply a significant reduction of design time and costs. There exist a few LS-based wet etching simulators but they have many limitations and they have never been validated with real experiments. On the other hand, atomistic models are currently considered the most advanced simulators. Nevertheless, atomistic simulators present some drawbacks like the requirement of a prior calibration process in order to use the experimental data. Additionally, a lot of effort must be invested to create an atomistic model for simulating the etching process of substrate materials with different atomistic structures. Furthermore, the final result is always formed by unconnected atoms, which makes difficult a

proper visualization and understanding of complex structures, thus, usually an additional visualization technique must be employed.

For its part, dry etching simulators usually employ an explicit representation technique to evolve the surface being etched according to etching models. This strategy can produce unrealistic results, especially in complex situations like the interaction of multiple surfaces. Despite some models that use implicit representation have been published, they have never been directly compared with real experiments and computational performance of the implementations have not been properly analysed.

The commented limitations are addressed in the various chapters of the present thesis, producing the following contributions:

- An efficient LS implementation in order to improve the visual representation of atomistic wet etching simulators. This implementation produces continuous surfaces from atomistic results.
- Definition of a new LS-based model which can directly use experimental data of many etchant solutions (such as KOH, TMAH, NH_4HF_2, and IPA and Triton additives) to simulate wet etching processes of various substrate materials (e.g. silicon and quartz).
- Validation of the developed wet etching simulator by comparing it to experimental and atomistic simulator results.
- Implementation of a LS-based tool which evolves the surface being etched according to dry etching models in order to enable the simulation of complex processes. This implementation is also validated experimentally.
- Acceleration of the developed wet and dry etching simulators by using Graphics Processing Units (GPUs).

In this thesis, all the listed solutions require the definition of thorough studies and methodologies of design in order to develop efficient and correct implementations, including benchmark experiments against other existing models and/or experimental results.

Resumen

El tema principal de la presente tesis consiste en mejorar la simulación de los procesos de fabricación utilizando el método *Level Set* (LS). El LS es una técnica matemática utilizada para la evolución de frentes según un movimiento definido por unas leyes en concreto. La principal ventaja de este método es que el frente está embebido dentro de una función definida en una dimensión superior. De esta forma, actualizar dicha función en lugar del propio frente permite tratar de forma trivial situaciones complejas como la separación o la colisión de diversos frentes.

En concreto, este documento se centra en los procesos de atacado húmedo y seco, los cuales son ampliamente utilizados en el proceso de fabricación de Sistemas Micro-Electro-Mecánicos (MEMS, de sus siglas en inglés). Un MEMS es un sistema formado por elementos mecánicos, sensores, actuadores y electrónica. Estos dispositivos han ganado mucha popularidad en las últimas décadas y son utilizados en muchos campos de la industria como la seguridad automovilística, sensores de movimiento y teléfonos inteligentes.

El proceso de atacado húmedo consiste en eliminar de forma selectiva el material del sustrato (por ejemplo, silicio o cuarzo) con una solución líquida con el fin de formar una estructura específica. Éste es un proceso complejo pues el resultado de un determinado experimento depende de muchos factores, tales como la estructura cristalográfica del material, la solución atacante o su temperatura. De forma similar, los procesos de atacado seco son utilizados para eliminar el material del sustrato, sin embargo, se utilizan sustancias gaseosas en la fase de atacado.

En ambos casos, la utilización de un simulador capaz de predecir de forma precisa el resultado de un experimento concreto implicaría una reducción significativa del tiempo de diseño y de los costes. Existen unos pocos simuladores del proceso de atacado húmedo basados en el método LS, no obstante tienen muchas limitaciones y nunca han sido validados con experimentos reales. Por otro lado, los simuladores atomísticos son hoy en día considerados los simuladores más avanzados. Sin embargo, estos simuladores tienen algunos inconvenientes como la necesidad de un proceso de calibración previo para poder utilizar los datos experimentales. Además, debe invertirse mucho esfuerzo para crear un modelo

atomístico para la simulación de materiales de sustrato con distintas estructuras atomísticas. Asimismo, el resultado final siempre está formado por átomos inconexos que dificultan una correcta visualización y un correcto entendimiento de aquellas estructuras complejas, por tanto, normalmente debe emplearse una técnica adicional para la visualización de dichos resultados.

Por su parte, los simuladores del proceso de atacado seco normalmente utilizan técnicas de representación explícita para evolucionar, según los modelos de atacado, la superficie que está siendo atacada. Esta técnica puede producir resultados poco realistas, sobre todo en situaciones complejas como la interacción de múltiples superficies. A pesar de que unos pocos modelos son capaces de solventar estos problemas, nunca han sido comparados con experimentos reales ni el rendimiento computacional de las correspondientes implementaciones ha sido adecuadamente analizado.

Las expuestas limitaciones son abordadas en los distintos capítulos de la presente tesis y se han producido las siguientes contribuciones:

- Implementación eficiente del método LS para mejorar la representación visual de los simuladores atomísticos del proceso de atacado húmedo. Esta implementación produce una superficie continua a partir de los resultados atomísticos.

- Definición de un nuevo modelo basado en el LS que pueda usar directamente los datos experimentales de muchos atacantes (tales como KOH, TMAH, NH_4HF_2 y aditivos como IPA y Triton) para simular el proceso de atacado húmedo de diversos materiales de sustrato (por ejemplo silicio y cuarzo).

- Validación del simulador de atacado húmedo desarrollado comparándolo con resultados experimentales y con los de simuladores atomísticos.

- Implementación de una herramienta basada en el método LS que evolucione la superficie que está siendo atacada según los modelos de atacado seco para, de esta forma, habilitar la simulación de procesos complejos. Esta implementación también es validada experimentalmente.

- Aceleración de los simuladores desarrollados de los procesos de atacado seco y húmedo mediante Unidades de Procesado Gráfico (GPUs, de sus siglas en inglés).

En esta tesis, todas las soluciones listadas requieren la definición meticulosa de estudios y metodologías para el desarrollo de implementaciones eficientes y correctas. Además, se han evaluado los resultados comparándolos con modelos existentes y/o con resultados experimentales.

Resum

El tema principal de la present tesi consisteix en millorar la simulació de processos de fabricació mitjançant el mètode *Level Set* (LS). El LS és una tècnica matemàtica utilitzada per a l'evolució de fronts segons un moviment definit per unes lleis en concret. El principal avantatge d'aquest mètode és que el front està embegut dins d'una funció definida en una dimensió superior. D'aquesta forma, actualitzar la dita funció en lloc del propi front, permet tractar de forma trivial situacions complexes com la separació o la col·lisió de diversos fronts.

En concret, aquest document es centra en els processos d'atacat humit i sec, els quals són àmpliament utilitzats en el procés de fabricació de Sistemes Micro-Electro-Mecànics (MEMS, de les sigles en anglès). Un MEMS és un sistema format per elements mecànics, sensors, actuadors i electrònica. Aquests dispositius han guanyat molta popularitat en les últimes dècades i són utilitzats en molts camps de la indústria, com la seguretat automobilística, sensors de moviment i telèfons intel·ligents.

El procés d'atacat humit consisteix en eliminar de forma selectiva el material del substrat (per exemple, silici o quars) amb una solució líquida, amb la finalitat de formar una estructura específica. Aquest és un procés complex ja que el resultat d'un determinat experiment depèn de molts factors, com l'estructura cristal·logràfica del material, la solució atacant o la seva temperatura. De manera similar, els processos d'atacat sec son utilitzats per a eliminar el material del substrat, no obstant, s'utilitzen substàncies gasoses en la fase d'atacat.

En ambdós casos, la utilització d'un simulador capaç de predir de forma precisa el resultat d'un experiment en concret implicaria una reducció significativa del temps de disseny i dels costos. Existeixen uns pocs simuladors del procés d'atacat humit basats en el mètode LS, no obstant tenen moltes limitacions i mai han sigut validats amb experiments reals. Per la seva part, els simuladors atomístics tenen alguns inconvenients com la necessitat d'un procés de calibratge previ per a poder utilitzar les dades experimentals. A més, deu invertir-se molt d'esforç per crear un model atomístic per a la simulació de materials de substrat amb diferents estructures atomístiques. Així mateix, el resultat final sempre està format per àtoms inconnexos que dificulten una correcta visualització i un correcte enteniment d'aquelles estructures complexes, per tant, normalment deu emprar-se una tècnica addicional per a la visualització d'aquests resultats.

D'altra banda, els simuladors del procés d'atacat sec normalment utilitzen tècniques de representació explícita per evolucionar, segons els models d'atacat, la superfície que està sent atacada. Aquesta tècnica pot introduir resultats poc realistes, sobretot en situacions complexes com per exemple la interacció de múltiples superfícies. A pesar que uns pocs models son capaços de resoldre aquests problemes, mai han sigut comparats amb experiments reals ni tampoc el rendiment computacional de les corresponents implementacions ha sigut adequadament analitzat.

Les exposades limitacions son abordades en els diferents capítols de la present tesi i s'han produït les següents contribucions:

- Implementació eficient del mètode LS per millorar la representació visual dels simuladors atomístics del procés d'atacat humit. Aquesta implementació produeix una superfície contínua a partir dels resultats atomístics.
- Definició d'un nou model basat en el mètode LS que puga utilitzar directament les dades experimentals de molts atacants (com ara KOH, TMAH, NH_4HF_2 i additius com IPA i Triton) per a simular el procés d'atacat humit de diversos materials de substrat (per exemple silici i quars).
- Validació del simulador d'atacat humit desenvolupat comparant-lo amb resultats experimentals i amb els de simuladors atomístics.
- Implementació d'una ferramenta basada en el mètode LS que evolucione la superfície que està sent atacada segons els models d'atacat sec per, d'aquesta forma, habilitar la simulació de processos complexos. Aquesta implementació també és validada experimentalment.
- Acceleració dels simuladors desenvolupats dels processos d'atacat sec i humit mitjançant Unitats de Processat Gràfic (GPUs per les segues sigles en anglès).

En aquesta tesi, totes les solucions llistades requereixen la definició meticulosa d'estudis i metodologies per al desenvolupament d'implementacions eficients i correctes. A més, s'han avaluat tots els resultats comparant-los amb models existents i/o amb resultats experimentals.

Acronyms

AFM	Atomic Force Microscopy. 52, 164, 166, 217
ALU	Arithmetic Logic Unit. 44
API	Application Programming Interface. 40–42
CA	Cellular Automata. 69, 70, 74, 78, 205, 208
CCA	Continuous Cellular Automata. 70, 77–80, 84, 95–97, 101–106, 109–115, 129, 131–133, 142, 153–174, 205–208, 214–217
CCP	Capacitively Coupled Plasma. 71
CFL	Courant-Friedrichs-Lewy. 23, 24, 28, 30, 83, 84, 126, 182
CPU	Central Processing Unit. 38–41, 43, 45, 49, 111, 115, 142–146, 148–150, 153, 156, 158, 159, 171–173, 189, 190, 194, 198, 202–204, 206–208, 212, 216, 217, 221
CSP	Closest S Point. 86, 88
CTM	Close To Metal. 42
CTS	Constant-Time-Stepping. 153–155, 159, 163, 216
CUDA	Compute Unified Device Architecture. 43–47, 50, 51, 144, 146, 147, 149, 153, 154, 163, 188–190, 192, 198, 202–204, 212, 221
CVD	Chemical Vapor Deposition. 54
DRIE	Deep Reactive Ion Etching. 58, 59, 74, 103, 106, 121, 210, 213
EA	Evolutionary Algorithm. 113, 114
ED	Electro-Deposition. 54
EDP	Ethylene Diamine Pyrocatechol. 61
ENO	Essentially Non Oscillatory. 24–26
FCC	Face Centered Cubic. 62, 63, 213
FMA	fused multiply-add. 50, 51
FPGA	Field-Programmable Gate Array. 39, 43
FPU	Floating-Point Unit. 44

GPGPU	General-Purpose computing on Graphics Processing Unit. 38–40, 42, 43
GPU	Graphics Processing Unit. 38–46, 49–51, 79, 111, 115, 142, 144, 146–153, 156, 158, 159, 163, 170–173, 175–177, 188–195, 198, 202–210, 212, 216–219, 221, 222, 224, 225
H-J	Hamilton-Jacobi. 15, 21, 24–26
ICP	Inductively Coupled Plasma. 58
IPA	Isopropyl alcohol. 65, 112, 113, 115, 121, 131, 133, 173, 206, 215
KMC	Kinetic Monte Carlo. 69, 70
KOH	potassium hydroxide. 61, 65, 67, 69, 73, 101, 103, 104, 106, 112, 113, 115, 119, 131–134, 153, 154, 156, 157, 165–167, 169, 173, 206, 213, 215–217
LFS	Lax-Friedrichs Schemes. 26, 28, 30, 119, 122, 125, 126
LLF	Local Lax-Friedrichs. 26
LS	Level Set. 7, 11, 14–16, 19–21, 24, 28–32, 34, 35, 37, 38, 51, 60, 71, 74, 75, 77, 79, 80, 82, 84, 88–92, 95–111, 114–116, 118, 119, 121–126, 128, 132, 133, 135, 142, 146, 147, 155, 171, 172, 175–177, 181–187, 189, 194, 197, 198, 200, 202–210, 212, 214, 215, 218, 221, 223
MEMS	Micro-Electro-Mechanical Systems. 51–55, 58, 61, 69, 73, 78, 79, 87, 110–112, 114, 116, 135, 136, 144, 153, 159, 163, 171, 172, 174, 176, 204, 205, 210, 212
NBM	Narrow Band Method. 10, 20, 21, 30–33, 212, 223
NEMS	Nano-Electro-Mechanical Systems. 51
openCL	Open Computing Language. 42, 43
Pthreads	POSIX Threads. 143, 144, 153
PVD	Physical Vapor Deposition. 55
R-K	Runge-Kutta. 24
RF	Radio Frequency. 51, 52, 55, 60, 71, 72
RIE	Reactive Ion Etching. 58, 59, 71–75, 176, 177, 190, 195, 199, 200, 203, 205, 207–209, 213, 214, 218, 219, 222
RLE	Run-Length Encoding. 37, 74

SDF	Signed Distance Function. 16, 19–21, 28–30, 32, 34, 35, 80, 83, 87–96, 107, 116, 118, 125, 128, 129, 149, 181, 185–187, 189–191, 194, 214, 215, 223, 224
SDK	Software Development Kit. 43
SFM	Sparse Field Method. 10, 21, 32–35, 37, 38, 74, 77, 79, 88, 91, 93–111, 115, 122, 124, 125, 127–130, 132, 134–137, 140–165, 167–175, 177, 182, 185–188, 190–195, 198, 200, 202–209, 212, 214–219, 221–225
SIMD	Single Instruction Multiple Data. 142
SM	Streaming Multiprocessors. 44, 45, 49, 50, 147
SSE	Streaming SIMD Extensions. 142
TMAH	tetramethylammonium hydroxide. 61, 65, 67, 112, 113, 115, 119, 131–134, 153, 165, 168, 173, 206, 215, 216
TPC	Thread Processing Cluster. 44
TVD R-K	Total Variation Diminishing R-K. 24, 25
UC	Unit Cell. 63, 96, 101, 102, 104, 105, 153–155, 160, 161, 165–169, 171
UV	Ultraviolet. 55
VOF	Volume Of Fluid. 8–10, 211
VPE	Vapor Phase Epitaxy. 54
WENO	Weighted ENO. 24–26, 83, 90, 214

Contents

Chapter 1

Introduction

The reasons that have motivated all the research effort invested during this thesis are compiled in this chapter, including the objectives established when defining the research tasks of the thesis. After this, the work and research strategies used in the different parts of the thesis are commented. Finally, the structure of this document is presented, summarizing the main contributions of every chapter.

1.1 Motivation

The Level Set (LS) method is a mathematical approach to evolve fronts according to certain motions. The essential idea is to embed the front in a one-higher-dimensional function such that the motion is applied to this function instead of the front itself, i.e. an implicit representation of the front. This enables a trivial handling of complicated situations like the splitting and coalescing of several fronts. The LS method has been applied in many fields, but this thesis is focused on improving the simulation of micromachining processes.

Current technological devices like smartphones and vehicles include miniaturized electronic and mechanical systems which tend to be smaller every year. There are several processes employed in the fabrication of these Micro-Electro-Mechanical Systems (MEMS), such as wet and dry etching processes.

Wet etching process consists in introducing a substrate inside an etchant solution which removes the substrate material. The behaviour of this removal process depends on many factors, such as: the mask pattern applied to avoid etching certain areas of the substrate, etchant solution, its concentration and temperature, substrate material, and its crystallographic orientation. The most used substrate material is silicon since it allows the integration with typical electronic components and a lot of effort has been invested in last decades in order to well understand the silicon wet etching process. This enables the possibility of creating complex three-dimensional structures by selecting properly all the experiment parameters.

Due to the high complexity of wet etching process, the prediction of the result of a specific experiment is very useful since it can reduce time and costs when designing a certain microstructure. Accordingly, several approaches have been developed to simulate silicon wet etching process. Current atomistic simulators based on Cellular Automata (CA) can describe accurately many silicon etchants and even some solutions for etching quartz substrates. Nevertheless, an atomistic simulator result is a cloud of unconnected points that makes difficult the well understanding and study of the resulting structures and, usually, an additional visualization method is required. In addition, CA-based simulators require the knowledge of the atomic structure of the material being etched as well as an exhaustive classification of the different atomistic configurations. Moreover, a computationally expensive calibration process must be performed each time that experimental conditions are changed, which can take several tens of hours.

As the technology evolves, new etchant solutions and materials are employed and more complex structures can be implemented. Thus, we think that the LS method can be used to improve the visualization of the CA results and also it can be a good alternative method which avoids calibration processes and it is independent on atomistic structure of the substrate material. Similarly, the implicit representation of the LS method could be used to improve the current dry etching simulators that cannot produce realistic results because of the employed explicit evolution front techniques required by dry etching models.

Furthermore, the Sparse Field Method (SFM) optimization reduces the computational time of the original LS method by only considering the space strictly close to the front. This optimization can be combined with the parallel nature of the LS method, which enables the possibility of reducing drastically the execution times by running the algorithms on parallel platforms like Graphics Processing Units (GPUs) or modern Central Processing Units (CPUs), influencing significantly the design of MEMS.

1.2 Objectives

The present thesis is focused on the theoretical and practical implementation of the LS method in order to improve the simulation of micromachining processes. The objectives imposed for this thesis are:

- **To validate the LS method as a tool for improving atomistic results of wet etching simulations.** Since atomistic methods produce a cloud of unconnected points, results visualization is difficult specially for complex structures. Hence, we want to take advantage of the LS properties in order to construct a smooth and continuous surface from the atomistic resulting points, independently of the complexity of the resulting structure.

- **To develop a versatile wet etching simulator based on the LS method capable of using directly experimental data.** In this point, we want to adapt the LS method to employ directly an extensive experimental data set, thus, avoiding any calibration process or any required knowledge of atomistic structure of the material being etched. This simulator should be, ideally able to reproduce faithfully the experimental results for every etchant solution and substrate material.

- **To accelerate LS simulations by implementing a parallel algorithm of the SFM.** The main drawback of the LS method is the higher computational cost due to the front is embed in a higher-dimensional function. Nevertheless, this issue can be alleviated by executing the algorithm on a parallel architecture, such as modern multi-core CPUs or many-core GPUs. The combination of both optimizations (SFM and parallelization) can reduce drastically the execution time of LS algorithms.

- **To perform a thorough comparison with the most advanced atomistic simulators of wet etching process as well as to compare the results with the corresponding experimental ones.** Since CA-based simulators are currently accepted as the most accurate and efficient approach to simulate wet etching process, we believe that every new developed simulator must be compared to them in terms of accuracy and computational performance. Similarly, the results produced with the proposed simulator must be validated against experimental ones.

- **To improve the current dry etching simulators by using a LS implementation to evolve the front being etched.** Many current dry etching simulators use an explicit representation of the surface being etched, however, this representation technique requires additional programming effort to handle complex situations like surface coalescing or disjoint. Accordingly, the usage of the LS method to evolve the etched surface would enable a trivial handling of these situations, allowing the simulation of more complex processes.

All these objectives are addressed in the different chapters of the thesis.

1.3 Methodology

In order to solve the imposed objectives, a strict work methodology has been followed to guarantee an adequate process to address the corresponding problems.

The first stage of this methodology was to identify the problem to be solved. Then, a thorough information research has been performed in order to know if the specific problem has been already tried to solve and to get as much information as possible to determine if solving the problem would make a significant contribution to scientific community.

After this, a theoretical study was performed in order to design a new methodology and procedures that solve the problem or improve the current limitations.

Then, these new procedures were implemented to verify the good results. The algorithms proposed in this thesis have been implemented in Java since it allows a quick development to validate the designed procedures. Some of these algorithms were then implemented in C and CUDA C in order to efficiently execute them on a multi-core CPU and a massively parallel GPU, respectively. Both programming languages provide an efficient performance, thus, they were used to accelerate the simulations. In addition, Nvidia CUDA C has been chosen since it is a relatively matured platform with well documented manuals and guides. The GPUs utilized in this thesis were the Nvidia GeForce GTX 260, the GeForce GTX 560, and the GeForce GTX Titan. Furthermore, the extraction and visualization procedures of LS surfaces have been developed with Matlab programming language due to the available libraries that can be directly used for such purposes.

After the implementation of the algorithms, the results were analysed in terms of computational performance and validity of the results. This task usually was an iterative process in which the algorithms were debugged, corrected and optimized. Then, the efficacy of the proposed solutions was also analysed by comparing the results with other methods and/or experimental ones.

Finally, when the proposed solutions have been corrected and validated, the corresponding documentation such as scientific articles and the present thesis, were written.

1.3.1 Employed tools

During the realization of the thesis, several tools have been utilized in the different stages of the explained methodology. Accordingly, in the first stages, several web portals have been utilized to find related information, such as: sciencedirect, iopscience, and ieeexplore. Especially interesting is the searching tool Google scholar since it searches over many portals and websites and it can provide even PhD thesis that are not listed in any scientific journal.

When implementing Java algorithms, several version of the Integrated Development Environment (IDE) Eclipse have been employed, from 3.8 (Juno) to 4.4 (Luna), which use the 1.7 or 1.8 versions of the Java Development Kit (JDK). Furthermore, the codes written in C programming language have been developed on the Microsoft Visual Studio 2010 IDE. The same tool has been employed for implementing parallel CUDA C algorithms to be executed on Windows-based Operating Systems (OSs). The Nvidia CUDA toolkit 5.0 has been used with this tool. On the other hand, the Nsight Eclipse Edition released by Nvidia has been used for developing CUDA C algorithms on Ubuntu OS, in combination with the 6.5 version of the CUDA toolkit. Moreover, still in the implementation stage, the R2011b (7.13) and R2014a (8.3) version of the MATLAB tool have been used for the surface extraction and visualization processes.

The computational performance of the proposed algorithms was analysed introducing timers in the code itself. Nevertheless, besides the applications themselves, the Nvidia CUDA Visual Profiler provides very useful information about the performance of CUDA C applications being executed on an Nvidia GPU. Hence, using both strategies, the algorithms were optimized by modifying them until an efficient performance was achieved. Finally, the results were compared in terms of accuracy with other methods results. For wet etching process, the obtained results were compared to those obtained with the CA-based simulator Intellietch 2.22, developed by Intellisense-Corp. On the other hand, for dry etching simulations, the Anetch 0.7.5 version, developed by the Fraunhofer IISB, was used for obtaining the corresponding results.

1.4 Structure of the thesis

The present document is divided into a total of six chapters, compiling all the research carried out during the thesis.

Chapter 2 contains useful information of the topics covered in the thesis. This information includes many up to date references, forming a state of the art of the different topics, namely: theory and numerical techniques of the LS method, GPU devices utilized as massively parallel execution platforms, and an introduction to MEMS, including explanations of wet and dry etching processes.

Later, in chapter 3 the first objective is addressed, i.e. to improve the visualization of the structures obtained with CA-based simulators of wet etching process. First, the existing problem is commented and two different solutions are proposed. The algorithms of these solutions are explained in detail and finally both are compared with the CA results to appreciate the improvement. This chapter is also used to demonstrate the better performance of the SFM in comparison with the original LS method.

Chapter 4 covers the next three objectives. This chapter is focused on the simulation of wet etching process with the LS method. The methodology previously described is followed to solve the problems of atomistic simulators. Accordingly, a SFM algorithm is proposed to simulate wet etching process taking directly experimental data. Furthermore, in order to exploit the parallel nature of the LS method, two parallel versions of the proposed algorithm are implemented: one to be executed on a multi-core CPU and another adapted for a GPU device. Both implementations are compared in terms of computational performance. Finally the results obtained with the proposed SFM algorithm are compared to those obtained with a CA approach and to the experimental ones, proving the efficacy and efficiency of the developed implementation. Hence, solving the three corresponding objectives.

The improvement of the dry etching simulators is addressed in chapter 5. In the first place, the problem of current dry etching simulators like Anetch is identified and explained. After commenting previous related solutions, a solution based on the SFM method is proposed. Furthermore, a new parallel implementation of the proposed SFM algorithm is developed to be executed on modern GPUs in order to reduce the computational time. Accordingly, an additional algorithm is developed to obtain a complete dry etching simulator which fulfils the last objective.

Finally, chapter 6 compiles all the contributions of this thesis. Additionally, all the publications derived from this work are presented. Since there is room for further research, the future work is also commented in this section. After this, the lists of figures, tables and algorithms, as well as the bibliographic references are included at the end of the document.

Chapter 2

State of the art

2.1 Introduction to the Level Set method

The main topic of the Thesis is the Level Set (LS) method applied to model and improve the simulation of semiconductor fabrication processes, therefore this section introduces it and explains the relevancy obtained in the last years by this method.

2.1.1 Tracking moving interfaces techniques

Many physical phenomena can be described as a moving interface which separates two or more regions, including semiconductor manufacturing processes (etching, deposition, crystal growth, etc.), combustion processes and meteorology among others. There are different ways to characterize a moving interface. The simplest scenario is an interface represented by a function $y = f(x,t)$. However, many phenomena cannot be described by such functions, for instance a closed curve, since a single value of x may produce several y values. For simplicity, consider a one-dimensional closed curve $\Gamma(t)$ moving in a two dimensional space S as shown in Fig. 2.1. The movement of the interface is defined by the velocity field $F = (u, v)$. This F field is determined by the physical process being modelled and can depend on local properties, such as local curvature of the curve.

There are mainly three ways to track the interface:

- *Geometric approach.* Suppose that the interface is parametrized by the variable s such that the image of each front point s_i at each time step $n\Delta t$ (assuming a homogeneous time discretization) is a marker point (x_i^n, y_i^n) on

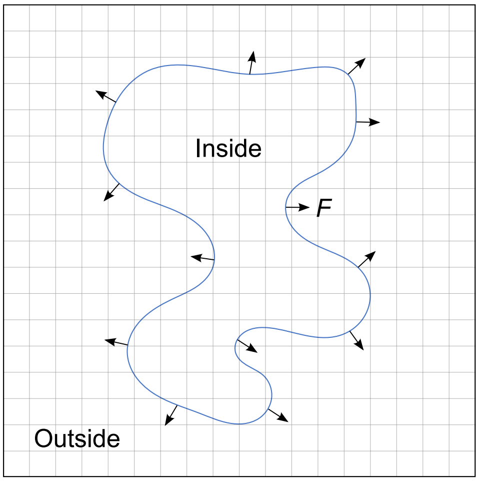


Figure 2.1: One-dimensional closed curve example. The front is determined by the interface of two regions. Local normal directions of the front are shown.

the moving front. An example of this is shown in Fig. 2.2. Therefore, unique values of the coordinates x and y are given by specific values of s and t, such that the front is defined as $\Gamma(t) = ((x(s,t), y(s,t))$. Accordingly, the equations of motion can by described by differentiating with respect to the parametrization variable s:

$$\begin{aligned} x_t &= u\left(\frac{y_s}{\sqrt{x_s^2+y_s^2}}\right), \\ y_t &= -v\left(\frac{x_s}{\sqrt{x_s^2+y_s^2}}\right), \end{aligned} \tag{2.1}$$

abandoning the underlying fixed coordinate system. This approach implies the discretization of the interface, thus special attention is required when different elements of the interface collide since collisions can create corners, cusps and changes in topology. Also, if the accuracy needs to be kept constant, new points must be added to the interface if it expands and some points of the interface must be removed when the size of it decreases.

- *Volume Of Fluid (VOF)*. Consider the function $f(x, y, t)$. This function represents the volume fraction of the material in a cell such that $f = 1$ inside the interface Γ, $f = 0$ outside and $0 < f < 1$ for the interface cells. An example of a volume fraction for the closed curve of Fig 2.1 is presented in Fig 2.3. Then, the motion of the volume fraction can be written as

$$f_t = F \cdot \nabla f. \tag{2.2}$$

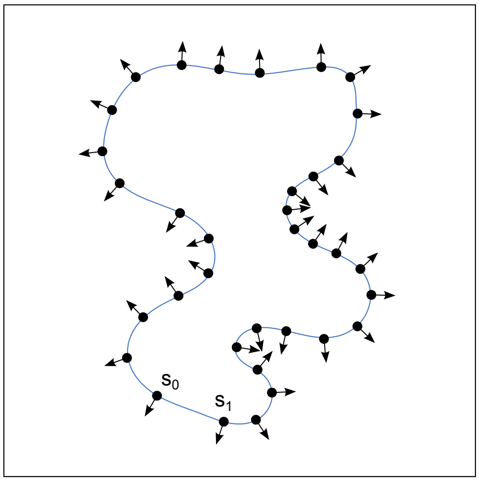


Figure 2.2: One-dimensional closed curve example parametrized by the s variable. The next position of each particle is pointed by the arrows.

According to this equation, all the points of the material are transported under the velocity field F. The advantages of this approach are the straight application in multiple dimensions, the ease of handling topological changes and the intrinsic nature of the VOF algorithms to conserve the mass of each fluid [1]. Nevertheless, since the function f is not continuous and it is discretized over a fixed underlying grid, (2.2) cannot be easily solved directly and an approximation of ∇f must be used to perform the evolution update. This is typically done through algorithms that reconstruct the front of the volume with different grades of accuracy [1–4].

- *Level Set method.* The third approach was presented by Osher and Sethian [5]. This method also uses a fixed grid over the space S. An implicit function $\phi(x, y, t)$ is defined over the whole space such that the zero level set $\phi(x, y, t) = 0$ corresponds to the evolving interface $\Gamma(t)$, i.e.:

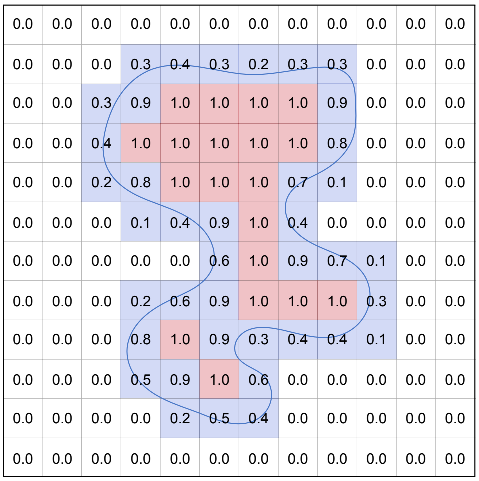
$$\phi(\Gamma(t), t) = 0. \tag{2.3}$$

The main idea of this method is to apply the motion to the implicit function ϕ, such that it would be possible to know ϕ at any time t with the motion equation $\frac{\partial \phi}{\partial t}$. In order to connect the front motion to the implicit function dynamics, the time derivative is applied to (2.3) and, according to the chain rule, it yields to

$$\phi_t + \nabla\phi(\Gamma(t), t) \cdot \Gamma'(t) = 0, \tag{2.4}$$

where $F = \Gamma'(t)$. Thus, the equation of the ϕ evolution can be written as

$$\phi_t + F \cdot \nabla\phi = 0. \tag{2.5}$$



0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.3	0.4	0.3	0.2	0.3	0.3	0.0	0.0	0.0
0.0	0.0	0.3	0.9	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.0
0.0	0.0	0.4	1.0	1.0	1.0	1.0	1.0	0.8	0.0	0.0	0.0
0.0	0.0	0.2	0.8	1.0	1.0	1.0	0.7	0.1	0.0	0.0	0.0
0.0	0.0	0.0	0.1	0.4	0.9	1.0	0.4	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.9	0.7	0.1	0.0	0.0
0.0	0.0	0.0	0.2	0.6	0.9	1.0	1.0	1.0	0.3	0.0	0.0
0.0	0.0	0.0	0.8	1.0	0.9	0.3	0.4	0.4	0.1	0.0	0.0
0.0	0.0	0.0	0.5	0.9	1.0	0.6	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.2	0.5	0.4	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 2.3: Volume fraction of a closed curve used in the VOF method. The blue cells represent the interface cells with $0 < f < 1$ and the red/white ones correspond to the cells that are completely inside/outside the front.

The advantages of this approach are the direct extension to multiple dimensions and the trivial handling of topological changes. Additionally, since function ϕ is defined over the whole space S and it is smooth, gradients and geometric characteristics such as local normal vectors and local curvatures can be calculated straightforwardly. On the other hand, the usage of an additional space dimension to build the implicit function implies a high computational cost $O(N^{n+1})$, where n is the number of dimensions of the interface, and N is the number of grid points in each dimension. Nevertheless, this can be alleviated with the Narrow Band Method (NBM), which updates only the points located within k layers counted from the front $\Gamma(t)$ itself, thus reducing the computational cost to $O(kN^n)$ [6]. However, this approach is not completely optimal because more points than the strictly necessaries are still updated and, in addition, the implicit function ϕ needs to be rebuilt periodically every time the front reaches the k-th layer. As a result, the Sparse Field Method (SFM) was introduced, which reduces the active region to only the strictly necessary points while efficiently updating the ϕ function in each time step [7].

Each of these three approaches has its own advantages and drawbacks and all of them are still currently used and contributing to each other. For instance, an implementation of the geometric approach, also known as front tracking or marker particle method, has been developed recently for fluid interfaces in compressible flows [8], as well as an algorithm to simulate solidification with volume change of a droplet on a fixed cooling plate [9]. The VOF method is still used for many

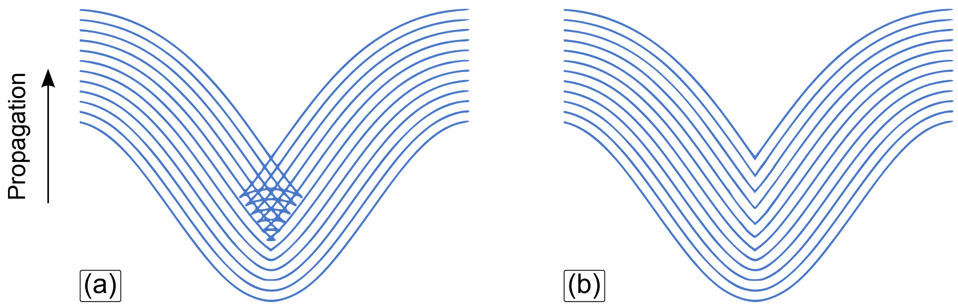


Figure 2.4: Cosine curve propagating with velocity $F = 1$. (a) Swallowtail solution, (b) entropy solution. Figure adapted by author from [25].

applications too, such as the simulation of wall to liquid heat transfer [10] or the simulation of electrohydrodynamic two-phase flows [11]. Similarly, the LS method has been used in many fields, such as image segmentation [12, 13], inverse problems [14, 15], chemical etching [16–21] and surface reconstruction from scattered points [22, 23].

2.1.2 Theory for front propagation

In the LS method, one of the main difficulties in solving the front evolution equation to emulate properly physical phenomena is that the front must be capable to present sharp changes in its topology, i.e. the solution needs to be nondifferentiable even with an initial smooth front. Therefore, accurate and efficient numerical techniques with the capability of producing this nondifferentiability are required.

In order to explain these requirements a simple example is commented [24]. Consider an initial front defined by a cosine periodic curve which is propagated in its normal direction with a given speed F. For simplicity, $F = 1$ is taken. As can be shown in Fig. 2.4(a), the front passes through itself and this solution cannot reproduce physical phenomena in which the front represents the boundary between two regions. This solution is known as double-valued swallowtail.

To properly represent a physical interface separating two regions, the front at time t should only consist of the set of all points located a distance t from the initial front. One way to build this solution is understanding the front like a propagating flame, such that once one grid point has been burnt, it cannot be burnt again [24]. This approach produces the result shown in Fig. 2.4(b) and satisfies the entropy condition of Huygens' Principle, i.e. no new information can be created during the evolution of the front. This means that once the entropy condition is invoke,

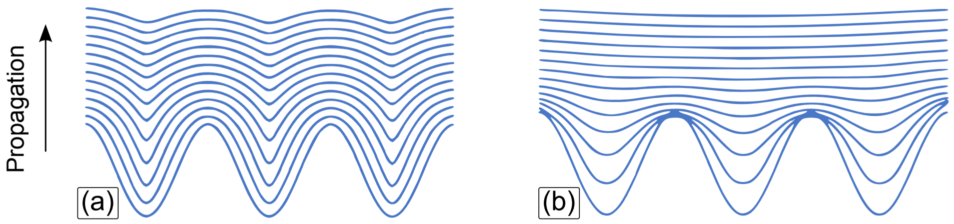


Figure 2.5: Triple sine curve propagating with velocity $F = 1 - \epsilon\kappa$. (a) $\epsilon = 0.025$, (b) $\epsilon = 0.25$. Figure adapted by author from [26].

i.e. the front reaches a burnt point, some information is lost and it is not possible to construct the previous front.

Fulfilling the entropy condition, a nondifferentiable weak solution is obtained after the occurrence of the singularity. Another way of obtaining this weak solution is through the limit of curvature-dependent propagating fronts [24]. Consider now a velocity function $F = 1 - \epsilon\kappa$, where ϵ is a constant and κ is the curvature. The curvature evolution equation can be written as [24]:

$$\kappa_t = \epsilon\kappa_{\alpha\alpha} + \epsilon\kappa^3 - \kappa^2, \tag{2.6}$$

where α is the arc length and the second derivative of the curvature κ is taken with respect to it. Eq. (2.6) is a reaction diffusion equation because of the reaction term $(\epsilon\kappa^3 - \kappa^2)$, which drives the front toward singularities, and the diffusion term $(\epsilon\kappa_{\alpha\alpha})$ that balance this effect by smoothing the front.

If the speed function $F(\kappa) = 1 - \epsilon\kappa, \epsilon > 0$ is applied to the cosine front, the front is sharpened at the trough by the negative reaction term since the curvature is negative $\kappa < 0$ at such points but it is smoothed by the positive diffusion term. Two cases with $\epsilon > 0$ are shown in Fig. 2.5 for a better understanding. The first case corresponds to a small value of ϵ and can be observed that the troughs sharpen up at the beginning of propagation. However, in the second case where the value of ϵ is larger, the parts of the front with high and positives curvature values move downward while the concave parts of the front move quickly up. Thus, for $\epsilon > 0$ the front stays smooth as shown in Fig. 2.6(a). On the other hand, with $\epsilon = 0$ a pure reaction equation is obtained $\kappa_t = -\kappa^2$ and a corner is developed in the exact solution $\kappa(s,t) = \frac{\kappa(s,0)}{1+t\kappa(s,0)}$, if the initial curvature is negative at some point as can be visualized in Fig 2.6(b).

As conclusion, two cases have been studied: (i) the front $\Gamma_{curvature}(t)$ with a curvature-dependent velocity $F = 1 - \epsilon\kappa, \epsilon > 0$, and (ii) the velocity constant front $\Gamma_{constant}(t)$ with $F = 1$. Then, at any time T

$$\lim_{\epsilon \to 0} \Gamma_{curvature}(T) = \Gamma_{constant}(T). \tag{2.7}$$

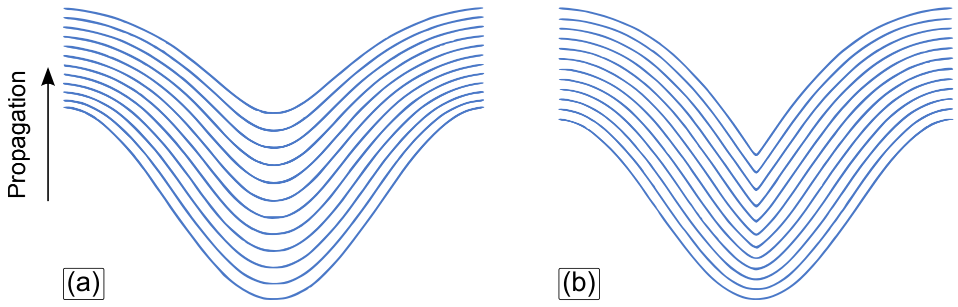


Figure 2.6: Cosine curve propagating with velocity $F = 1 - \epsilon\kappa$. (a) $\epsilon = 0.25$, (b) $\epsilon = 0$. Figure adapted by author from [25].

Thus, the limit of the front with curvature-dependent velocity corresponds to the entropy solution for the constant velocity front.

This limit is named the *viscous limit* because of its similarity with a viscous *hyperbolic conservation law*, which is defined for $u(x,t)$ by an equation of the form

$$u_t + [G(u)]_x = 0. \tag{2.8}$$

A simple example is given by the Burgers' equation:

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \tag{2.9}$$

which describes the motion of a compressible fluid in one dimension. The solution of this equation can develop discontinuities, known as shocks, even for arbitrarily smooth initial data. Nevertheless, the development of these shocks can be stop if a viscosity term is added in the equation as a diffusive term on the right side:

$$u_t + \left(\frac{u^2}{2}\right)_x = \epsilon u_{xx}. \tag{2.10}$$

For $\epsilon > 0$ can be demonstrated that solution must remain smooth for all the time. The role of curvature in a propagating front is analogous to the role of viscosity in the hyperbolic conservation law. In fact, the previously commented entropy condition is equivalent to the condition that enables that a compressible viscous fluid suddenly expands or compresses [24].

Consider an initial front given by the graph $f(x)$ with f and f' periodic on $[0,1]$, that remains as a graph for all time. Define ψ as the height of the propagating front at time t such that $\psi(x, t = 0) = f(x)$. According to Fig. 2.7 the change in height V in a unit time is related to the speed F in the normal direction by:

$$V = F\frac{\sqrt{1+\psi_x^2}}{1}, \tag{2.11}$$

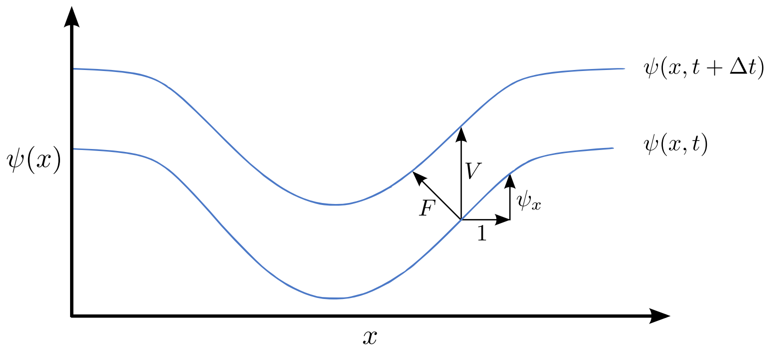


Figure 2.7: Representation of an initial graph $\psi(x,t)$ and the updated one $\psi(x,t+\Delta t)$ according to the normal velocity F. Figure adapted by author from [26].

being the equation of motion

$$\psi_t = F\sqrt{1+\psi_x^2}. \tag{2.12}$$

If the velocity function $F(\kappa) = 1 - \epsilon\kappa$, and the definition for the curvature $\kappa = -\psi_{xx}/(1+\psi_x^2)^{3/2}$ are used, then:

$$\psi_t - \sqrt{1+\psi_x^2} = \epsilon\frac{\psi_{xx}}{1+\psi_x^2}. \tag{2.13}$$

Differentiation of both sides of this equation yields an evolution equation for the slope $u = \partial\psi/\partial x$ of the propagating front such that,

$$u_t + \left[-\sqrt{1+u^2}\right]_x = \epsilon\left[\frac{u_x}{1+u^2}\right]_x. \tag{2.14}$$

This equation, i.e. the derivative of the curvature-modified equation for the changing height ψ of a graph, is similar to viscous hyperbolic conservation law (2.10) for the propagating slope u. This shows the relation between curvature-dependent moving fronts and the hyperbolic conservation law [27].

2.1.3 Similarity with Hamilton-Jacobi equations

The relation of the moving fronts with the hyperbolic conservation law previously explained is relevant because hyperbolic equations have been well studied and numerical schemes already developed can be applied to front propagation theory. Consider now the LS view that the front is embedded in a higher dimensional function

$$\phi_t + \vec{F}\cdot\nabla\phi = 0, \tag{2.15}$$

or, if the motion is in the normal direction,

$$\phi_t + F\,|\nabla\phi| = 0, \tag{2.16}$$

where $|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2}$. The hyperbolic conservation law cannot be applied directly to this LS formulation due to the multiple variable partial differentiation. To solve this, the similarity with Hamilton-Jacobi (H-J) equations was studied. These partial differential equations are usually used in mechanics and enable to obtain the temporal evolution equations of a wave or a particle. Consider the general H-J equation

$$\phi_t + H(\nabla\phi) = 0. \tag{2.17}$$

The function H is known as *Hamiltonian*. This equation is equivalent to the LS equation where $H(\nabla\phi) = \vec{F}\cdot\nabla\phi$ or in the normal motion $H(\nabla\phi) = F\,|\nabla\phi|$, but only if the velocity F depends exclusively on position x and first derivatives of ϕ. However, equations for curvature-dependent motion depends on the second derivatives of ϕ and they are not H-J equation type.

Considering the one-dimensional H-J equation

$$\phi_t + H(\phi_x) = 0 \tag{2.18}$$

and taking spatial derivatives of the entire equation, it becomes

$$(\phi_x)_t + H(\phi_x)_x = 0. \tag{2.19}$$

Therefore, setting $u = \phi_x$ results in

$$u_t + H(u)_x = 0, \tag{2.20}$$

which is equivalent to (2.8), thus the link in one-dimension between H-J equations and hyperbolic conservation law is proved. Accordingly, the usage of viscosity solutions of H-J equations were proposed [28, 29]. This was culminated by proposing a general framework for the numerical solution of H-J equations using successful methods from the theory of conservation laws [30–32]. The extension of these methods was possible since H-J equations in one spatial dimension are integrals of conservation laws [5]. More information about the links between the LS formulation, hyperbolic conservation laws, and H-J equations can be found in literature [26, 33].

2.1.4 Implicit functions

In the last sections the links between the LS method and hyperbolic conservation laws as well as H-J equations have been exposed. These similarities are used for building the adequate numerical techniques for propagating fronts that enable the capability to produce singularities like nondifferentiable curves. Depending on features of the motion, different numerical techniques must be used. This is discussed in section 2.2.

The main idea of the LS method is to encapsulate the front Γ in a higher dimensional implicit function ϕ such that the front always corresponds to a particular level of the implicit function. Typically the zero level is chosen, thus $\phi(\Gamma) = 0$. Unlike the marker particle methods (see section 2.1.1) which parametrize the front, in the LS method the implicit function is used for representing and evolving the front.

Consider a one-dimensional front Γ moving in a two dimensional space and an implicit function $\phi(x, y)$ defined over the whole space, such that the front corresponds to the zero level of ϕ, i.e. the implicit definition of the front is:

$$\phi(x, y) = 0. \tag{2.21}$$

Thus, every point $p = (x_0, y_0)$ that satisfies the condition (2.21) forms part of the front Γ. However, if $\phi(p) < 0$, the point p is inside the front and if $\phi(p) > 0$ the point is outside the front. Thus, an implicit function only can represent a front that separates the space into subdomains with nonzero areas such as spheres or planes.

Consider the one-dimensional example shown in Fig. 2.8. The simple closed curve is a circumference, defined by the equation

$$x^2 + y^2 - 5 = 0, \tag{2.22}$$

which corresponds to the zero level of the implicit function ϕ used to embed the circumference, such that

$$\phi(\Gamma) = x^2 + y^2 - 5 = 0. \tag{2.23}$$

The implicit function used in this example is the Signed Distance Function (SDF), and separates the space into two subdomains: the interior of the front with $\phi(x, y) < 0$ and the exterior of the circumference with $\phi(x, y) > 0$. To build the SDF, first a discretization of the two dimensional space has to be performed. In Fig. 2.8(a) the generated grid is shown. Then, the distance value between every grid point and the front Γ is calculated and stored with the corresponding sign depending if the grid point is outside (positive) or inside (negative). These distance values are shown in Fig. 2.8(a) with colours and, if they are represented as the third dimension, the implicit function SDF ϕ is formed as shown in Fig. 2.8(b). To discretize implicitly a front moving on $\Re^n$, an n-dimensional set of points needs to be resolved, whereas to discretize a parametric representation, it only needs to be resolved an $(n-1)$-dimensional set. This can be a drawback since it implies a higher computational cost, however, this can be partially avoided by only resolving those points very close to the front, leaving the rest unresolved.

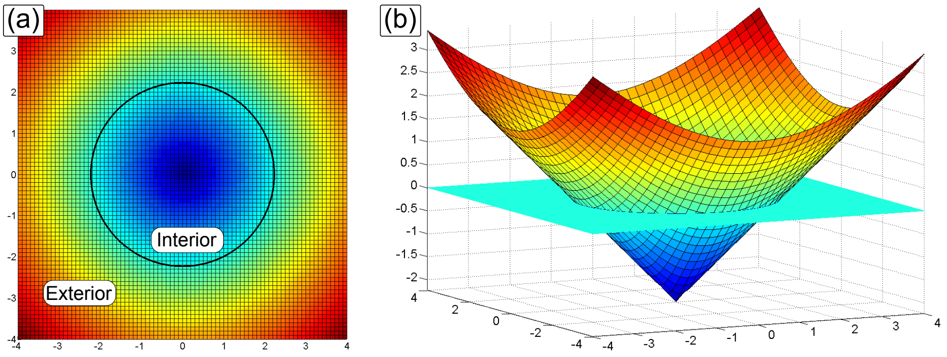


Figure 2.8: Implicit representation of a circumference. The signed distance function is used as implicit function to embed the front. (a) View from Z axis, the black line represent the circumference front and the two subdomains separated by it. (b) Three-dimensional representation of the implicit function. The zero level which contains the front is shown.

Another aspect to consider about implicit functions is the extraction of the implicit interface. In implicit representations, only values of ϕ at the generated grid cells are known, but usually almost none of them are exactly placed at the interface, i.e. exactly $\phi(x, y) = 0$. Thus, in order to locate the interface, isocontour needs to be interpolated from known values of ϕ at the grid points. This is a standard procedure accomplished by a variety of contour plotting routines. One of the most used algorithms for extracting isosurfaces moving on three-dimensional spaces is the so-called *marching cubes* method [34]. Basically, this method takes eight neighbouring cells for each grid cell (forming a cube) and determines the polygons necessaries to represent the part of the surface that goes through this cube. Finally all the polygons are joined and the surface is formed. Nevertheless, new algorithms for the same purpose have been presented [35].

Due to the required discretization of the space and the consequent interpolation to extract the implicit front, an error is introduced. This error is reduced if the number of grid cells is increased, reaching zero error for infinite grid cells. Nevertheless, numerical techniques and conditions can guarantee stability and a valid result. This is discussed in section 2.2, but generally if the implicit function is smooth enough over the grid, these estimates will be appropriate. For example, the gradient of an implicit function defined over a three-dimensional space is

$$\nabla\phi = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right). \tag{2.24}$$

The gradient vector is perpendicular to isosurfaces and points in the direction of increasing ϕ, i.e. outward. Thus, the unit normal vector is defined as

$$\vec{N} = \frac{\nabla \phi}{|\nabla \phi|}. \tag{2.25}$$

This definition of the normal vector is valid not only on the interface but over the whole space.

Since the space is discretized with a grid, the derivatives used for calculating the gradient need to be approximated using, for example, finite difference techniques. Only the equations of x variable are shown, but for the rest of variables is analogous. Different grades of accuracy can be obtained. For instance, a first-order accurate forward difference:

$$\frac{\partial \phi}{\partial x} \approx \phi_x^+ = \frac{\phi_{i+1} - \phi_i}{\Delta x}, \tag{2.26}$$

where Δx is the grid resolution used in the x dimension and the subscript i is used for referencing a specific grid point in the x dimension. The first-order backward difference is defined as:

$$\frac{\partial \phi}{\partial x} \approx \phi_x^- = \frac{\phi_i - \phi_{i-1}}{\Delta x}. \tag{2.27}$$

A second-order accurate central difference can be calculated with:

$$\frac{\partial \phi}{\partial x} \approx \phi_x = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \tag{2.28}$$

Higher order accurate differences can be calculated but generally require a higher computational cost [31, 32].

The mean curvature of the interface is defined as the divergence of the normal vector $\vec{N} = (n_1, n_2, n_3)$ such that,

$$\kappa = \nabla \cdot \vec{N} = \frac{\partial n_1}{\partial x} + \frac{\partial n_2}{\partial y} + \frac{\partial n_3}{\partial z}, \tag{2.29}$$

thus, $\kappa > 0$ for convex regions of the front, $\kappa < 0$ for concave ones and $\kappa = 0$ for planes. This equation can be directly written as a function of partial derivatives of ϕ since

$$\kappa = \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right), \tag{2.30}$$

leading to

$$\begin{aligned} \kappa = (&\phi_x^2 \phi_{yy} + \phi_y^2 \phi_{xx} + \phi_x^2 \phi_{zz} + \phi_z^2 \phi_{xx} + \phi_y^2 \phi_{zz} + \phi_z^2 \phi_{yy} \\ &-2\phi_x \phi_y \phi_{xy} - 2\phi_x \phi_z \phi_{xz} - 2\phi_y \phi_z \phi_{yz}) / |\nabla \phi|^3, \end{aligned} \tag{2.31}$$

which is in terms of the first and second order derivatives of ϕ. The lasts can be calculated with a second-order accurate finite difference

$$\begin{aligned} \phi_{xx} &= \frac{\partial^2 \phi}{\partial x^2} \approx \frac{\phi_{i+1}-2\phi_i+\phi_{i-1}}{\Delta x^2} \\ \phi_{xy} = \phi_{yx} &= \frac{\partial^2 \phi}{\partial x \partial y} \approx \frac{\phi_{i+1,j+1}-\phi_{i-1,j+1}-\phi_{i+1,j-1}+\phi_{i-1,j-1}}{4\Delta x \Delta y} \end{aligned} \tag{2.32}$$

With these equation, the normal vector $\vec{N}$ and the curvature κ can be calculated for every grid point. Then, an approximation of the value of the normal vector $\vec{N}$ at an interface point $\vec{x}_0$ can be also obtained by interpolating the values of $\vec{N}$ from the spatial neighbouring grid points to the point $\vec{x}_0$. This interpolation procedure requires ϕ to be smooth and well behaved. A good choice that satisfies this requirements is the SDF [33].

2.1.5 Signed Distance Function

In the LS method an implicit function is used for encapsulating and embedding a front. Thus, the front separates two regions of the space, acting as an interface. The distance function for $\vec{x}$ points is defined as:

$$d(\vec{x}) = min(|\vec{x} - \vec{x}_{in}|), \tag{2.33}$$

where $\vec{x}_{in}$ represents the interface points. Initially, the function $\phi(\vec{x}) = 1 \pm d^2(\vec{x})$ was used as implicit function in the LS method [5]. Nevertheless, the SDF proved that was a better choice [36]. A SDF is an implicit function ϕ with $|\phi(\vec{x})| = d(\vec{x})$ for every point $\vec{x}$. Thus, $\phi(\vec{x}) = d(\vec{x}) = 0$ for all $\vec{x}$ that belongs to the interface, $\phi(\vec{x}) = -d(\vec{x})$ for all points that are interior to the interface and $\phi(\vec{x}) = d(\vec{x})$ for exterior points. This function satisfies all the features explained in section 2.1.4. An SDF example of a two-dimensional front is shown in Fig. 2.8.

To build a distance function, for every grid point $\vec{x}$, its closest interface point has to be found. This closest point is labelled as $\vec{x}_c$. Then, the distance d between both points is calculated. Then, for every point $\vec{y}$ on the line segment connecting $\vec{x}$ and $\vec{x}_c$, $\vec{x}_c$ is the closest point for $\vec{y}$ as well. In fact, due to d is Euclidean distance, the next condition is fulfilled:

$$|\nabla d| = 1, \tag{2.34}$$

since a point that is twice as close to the interface, gives a value of d that is half as big. Eq. (2.34) is known as *Eikonal* equation, and it is satisfied by both distance function and SDF, i.e. $|\nabla \phi| = 1$. However, the argument exposed is only valid as long as only a unique closest point $\vec{x}_c$ exists for a point $\vec{x}$. Therefore, this approximation has to be used with caution to avoid overall degradations of the numerical method. According to this approximation, the normal vector results in

$$\vec{N} = \nabla \phi, \tag{2.35}$$

whereas the curvature (2.31) can be simplified to:

$$\kappa = \Delta\phi, \tag{2.36}$$

where $\Delta\phi$ is the Laplacian of ϕ such that,

$$\Delta\phi = \phi_{xx} + \phi_{yy} + \phi_{zz}. \tag{2.37}$$

These simplifications are relevant and can reduce computational effort, however, special attention must be taken since kinks generally have $|\nabla\phi| \neq 1$ and these equations can be insufficiently accurate.

2.1.6 Reinitialization of the implicit function

Once the implicit function ϕ is built, the LS method evolves this function according to a certain motion. Although this function is initialized as a SDF, it can be deteriorated and noise can be introduced due to the finite differences approximations. As interface evolves, ϕ will generally become a function that is no longer a SDF and, thus, the approximations of section 2.1.5 cannot be used. In addition, this can lead to produce instabilities and/or produce unrealistic results [37].

To avoid this troubles, a technique known as *reinitialization* was proposed [38]. This technique keeps ϕ as a SDF during the evolution of the front by replacing ϕ by another similar function $\tilde{\phi}$ that fulfils the conditions of a SDF. The new function $\tilde{\phi}$ keeps the interface in its zero level but is softer than ϕ and is better behaved. It was proved that this replacement is mathematically correct since the interface is kept in the same level zero of the function [39, 40].

This technique can be applied depending on the sensitivity of ϕ. If ϕ is very sensitive, it needs to be reinitialized to a SDF both accurately and often. Additionally, since ϕ can develop noisy and sharp features that are not suitable for finite difference approximations, it is always advisable to reinitialize ϕ occasionally so that it stays sufficiently smooth.

The straightforward way to reinitialize the function ϕ is to locate the front by interpolation techniques and calculate explicitly the exact SDF. This method requires a high computational effort specially if it is applied after every time step [41]. For this reason, more elaborated solutions were presented [42, 43]. These techniques avoid the need to locate the front and basically consist of solving a partial differential equation to steady state.

In order to reduce computational effort and get reasonable run times, a new method was proposed that restricted the calculations of motion and reinitialization to a small band of points close to interface $\phi = 0$ [38]. This technique is the first version of the *local* LS method and it was named as the NBM and analysed extensively [6]. However, the NBM is not optimal since more points than the strictly necessary are updated. To solve this, the SFM was proposed by Whitaker [7], which only updates the points that are necessaries to evolve the surface. Therefore, if only first-order accurate finite differences are required, only 3 layers of grid points are updated: the one containing the interface itself, those points at distance Δx from the front and the layer of points at distance $-\Delta x$ from the front. Additionally, another advantage of the SFM is that in each time step the SDF is built and, thus, there is no need to use any of these reinitialization techniques. These local LS methods are discussed in section 2.2.4.

Nevertheless, there are some applications where is useful to update more than one level of the implicit function and, perhaps, none of these techniques can be used. For that reason, nowadays new reinitialization techniques are still being presented [44–46].

2.2 Numerical schemes for the Level Set method

In section 2.1, the origin of the LS has been studied. The development of this method started with the analysis of corners and singularities in propagating interfaces. The roles of curvature and smoothing viscous term have shown the relation between entropy conditions and hyperbolic conservation laws. Similarly, the connection between LS equations and H-J equations was presented. The numerical schemes developed for approximating the solution to these equations can be used by the LS method to evolve the front. The basic evolution equation

$$\phi_t + \vec{V} \cdot \nabla \phi = 0, \tag{2.38}$$

depends on the velocity $\vec{V}$ and it is known as the *convection* equation. The implicit function ϕ contains the interface, thus, numerical methods can be applied to evolve it forward in time and, consequently, moving the interface across the grid. Consider that both ϕ and $\vec{V}$ are defined over the whole space which is discretized as commented in section 2.1.4 (i.e. building a grid)[1]. At a specific time t^n, let $\phi^n = \phi(t^n)$ represent the current values of ϕ. Updating ϕ in time consists in finding its new values $\phi^{n+1} = \phi(t^{n+1})$ at every grid point for the next time step $t^{n+1} = t^n + \Delta t$, where Δt is the time increment.

The velocity field can depend on different factors such as the position over the grid and local topology of the front. Regarding these features of the motion, different numerical techniques must be used. Following, three distinct motions and their corresponding numerical schemes are presented.

[1]This is not always possible and the *extension velocity* technique must be applied to extend the front velocity to the rest of the grid when it is not straightforward extendible [6, 47–49].

2.2.1 Completely external

Consider a velocity field defined over the whole space such that only the coordinates are needed to determine the value for a particular point. Additionally, this field can vary over time or not, such that at specific time t^n the velocity field is defined as:

$$\vec{V}^n = \vec{V}(\vec{x}, t). \tag{2.39}$$

By applying the simplest and first-order accurate method for the time discretization of (2.38), the *forward Euler* method, results in:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \vec{V}^n \cdot \nabla \phi^n = 0. \tag{2.40}$$

If this equation is extended for a three-dimensional case, it can be written as:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n \phi_x^n + v^n \phi_y^n + w^n \phi_z^n = 0, \tag{2.41}$$

where u, v, w are the components of $\vec{V}$ in dimensions x, y, z respectively. The numerical techniques can be applied independently for every dimensional term, thus, only the one-dimensional case is considered:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n \phi_x^n = 0. \tag{2.42}$$

The term u^n determine whether the front is moving to the right or to the left and it only depends on the grid point position. Thus, this equation must be solved for every grid point. Consider a certain grid point at position i such that

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + u_i^n (\phi_x)_i^n = 0, \tag{2.43}$$

where the subscript i indicates that the values correspond to the grid point x_i. The spatial derivative $(\phi_x)_i^n$ could be approximated in a straightforward manner by (2.26), (2.27) or (2.28), however the solution would diverge and appropriate approximations must be chosen adequately. For this reason, the upwind differencing or *upwinding* was presented [5]. The basic idea of this method is to take information from left if the front is moving to right and vice versa. Accordingly, if $u_i^n > 0$ at point x_i, the value of ϕ_i is moving from left to right, i.e. the left value ϕ_{i-1} will be transported to the position i at the end of a time step. Consequently, backward derivative ϕ_x^- should be used in this case. On the other hand, if $u_i^n < 0$, the ϕ_i value is moving from right to left, so information from the right side of x_i must be used for determining the value of ϕ_i at t^{n+1}. In this case, the forward derivative ϕ_x^+ must be used. Finally, if $u_i^n = 0$, the term $u_i^n (\phi_x)_i^n$ is also zero and there is no need to approximate ϕ_x.

The upwinding method for any time and any point is summarized in algorithm 1.

Algorithm 1: Upwind differencing technique

```
if $u > 0$ then
    $\phi_x$ is approximated by $\phi_x^-$.
else
    if $u < 0$ then
        $\phi_x$ is approximated by $\phi_x^+$.
    else
        The $u(\phi_x)$ term vanishes.
```

Since approximations of the derivatives (2.26) and (2.27) are first-order accurate, the errors are $O(\Delta x)$.

To obtain a convergent solution of (2.38), the result must be consistent and stable. The combination of the forward Euler time discretization with the upwind difference scheme is a consistent finite difference since the error converges to zero as $\Delta t \to 0$ and $\Delta x \to 0$. In addition, stability can be guaranteed by using the Courant-Friedrichs-Lewy (CFL) condition [50]. This condition avoids the error of the derivatives approximations to be amplified while the front is being evolved since it asserts that $\frac{\Delta x}{\Delta t} > |u|$, i.e. the numerical wave speed must be at least as fast as the physical wave speed. This turns into the CFL time step restriction:

$$\Delta t < \frac{\lambda \Delta x}{\max\{|u|\}}, \tag{2.44}$$

where $\max\{|u|\}$ is selected as the largest value of $|u|$ over the entire grid and λ is a scalar value such that $0 < \lambda < 1$. Usually, $\lambda = 0.9$ is chosen when a rapid convergence is needed, nevertheless, $\lambda = 0.5$ is a more conservative choice. The CFL condition can be written for multiple dimension, for example, for the three-dimensional case is:

$$\Delta t < \frac{\lambda}{\max\left\{\frac{|u|}{\Delta x} + \frac{|v|}{\Delta y} + \frac{|w|}{\Delta z}\right\}}, \tag{2.45}$$

where $\Delta x, \Delta y$ and Δz are the spatial resolutions of the corresponding dimensions.

2.2.1.1 Enhancing of the derivatives approximations

If the application requires more accuracy, instead of using first-order approximations of derivatives, more accurate approximations can be used, such as second-order central differences. However, these approximations are not compatible with first-order Euler discretization and they require a time step $\Delta t \sim \Delta x^2$, i.e. a more restrictive CFL condition, implying a high computational cost. Therefore, the Essentially Non Oscillatory (ENO) polynomial interpolation can be used in combination with upwinding and first-order Euler time discretization. This technique was first developed for hyperbolic conservation laws [30]. The basic idea of this method is to find the smoothest polynomial to approximate $\nabla\phi$. This method was improved significantly by the construction of $\nabla\phi$ directly from a divided difference table of the pointwise data [51, 52]. Finally, this technique was extended to the H-J equations, such as (2.38), taking advantage of the similarity between hyperbolic conservation laws and this type of equations [5]. This is known as H-J ENO and it allows to approximate ϕ_x^+ and ϕ_x^- with a third-order accurate by using a subset of $\{\phi_{i-3}, \phi_{i-2}, \phi_{i-1}, \phi_i, \phi_{i+1}, \phi_{i+2}, \phi_{i+3}\}$ values.

While the ENO method consists in finding the smoothest polynomial, an improved scheme known as Weighted ENO (WENO) was presented [32]. This method takes a convex combination of the different polynomial approximations produced by ENO. If any of the approximations interpolates across a discontinuity, it is given minimal weight in this combination to minimize the resulting errors. On the other hand, in smooth regions of $\nabla\phi$, all the polynomials have a significant weight so the local accuracy is improved from third-order to fourth-order. Furthermore, the WENO was optimized to produce a fifth-order accurate in smooth regions [53]. Similarly to the ENO method, the WENO scheme was extended to the H-J framework, enabling to solve (2.38) accurately. More information about this two techniques can be found in scientific literature [54]. Additionally, these methods are being continuously researched and new improvements are presented for hyperbolic conservation laws [55–57].

2.2.1.2 Enhancing the temporal discretization

Similarly to spatial derivatives which can be improved in terms of accuracy, the first-order forward Euler discretization of time used in (2.40) can be enhanced. Although the LS method is not usually so sensitive to temporal discretization as to spatial accuracy, in some cases an accurate temporal discretization may be required to obtain accurate numerical solutions.

The Runge-Kutta (R-K) numerical method was developed to discretize time to solve ordinary differential equations. Basically, this technique takes several Euler steps and combines the results with the initial data using a convex combination. The Total Variation Diminishing R-K (TVD R-K) method was presented and applied to the LS method [51] which can increase the accuracy of time discretization. The TVD R-K guarantees that no spurious oscillations are produced because of this accuracy increment in the time discretization, as long as no spurious oscillations are produced with the Euler steps. This technique, e.g. a third-order accurate TVD R-K method, can be used in combination with central differencing to obtain a numerical stable solution of (2.38). Accordingly, it is also possible to use a TVD R-K method together with upwind differences and H-J ENO or H-J WENO.

Although fourth-order and higher accurate TVD R-K schemes exist, they require much computational effort and the improvement of accuracy does not make a significant difference in practical calculations [51].

2.2.2 External and dependent on local topology

In section 2.2.1, the velocity field for a specific time t^n that corresponds to a particular point $\vec{x}_i$ is only determined by its coordinates (x_i, y_i, z_i), i.e. $\vec{V}(\vec{x}_i, t^n)$. Therefore, it does not depend on the local topology of the front. Now consider an external velocity field that depends on some local geometrical properties of the implicit function ϕ instead of only the grid coordinates of the point. For example, the velocity field can be obtained with the local normal vector $\vec{N}$, such that $\vec{V} = \vec{V}(\vec{N}, t)$. As before, $\vec{V}$ can vary over time or not. It is assumed that the velocity field $\vec{V}$ is defined for every grid point and the normal vector $\vec{N}$ can be calculated straightforwardly with (2.25) not only on the interface.

The case of an externally generated velocity field and this case, where the velocity field depends on ϕ, are both defined by (2.38) which is a H-J type equation like (2.17). Nevertheless, different numerical techniques need to be applied to each case due to the different dependencies of $\vec{V}$.

A function is considered convex if the *Hessian* matrix He is non-negative in all of its values. Consider a Hamiltonian that depends on multiple variables $H(\phi_{x_1}, \phi_{x_2}, \cdots, \phi_{x_n})$, then its He matrix is defined by

$$He(H) = \begin{pmatrix} \frac{\partial^2 H}{\partial \phi_{x_1}^2} & \frac{\partial^2 H}{\partial \phi_{x_1} \partial \phi_{x_2}} & \cdots & \frac{\partial^2 H}{\partial \phi_{x_1} \partial \phi_{x_n}} \\ \frac{\partial^2 H}{\partial \phi_{x_2} \partial \phi_{x_1}} & \frac{\partial^2 H}{\partial \phi_{x_2}^2} & \cdots & \frac{\partial^2 H}{\partial \phi_{x_2} \partial \phi_{x_n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 H}{\partial \phi_{x_n} \partial \phi_{x_1}} & \frac{\partial^2 H}{\partial \phi_{x_n} \partial \phi_{x_2}} & \cdots & \frac{\partial^2 H}{\partial \phi_{x_n}^2} \end{pmatrix}, \tag{2.46}$$

where, in a three-dimensional case, $x_1 = x, x_2 = y$ and $x_3 = z$. A compact form for expressing the condition for a convex Hamiltonian is

$$\frac{\partial^2 H}{\partial \phi_{x_i} \partial \phi_{x_j}} \geq 0, \tag{2.47}$$

for all the possible combinations of x_i, x_j. Condition (2.47) is fulfilled when velocity field $\vec{V}$ is completely externally generated. On the other hand, if $\vec{V}$ depends on the implicit function ϕ, this condition is usually not satisfied, i.e. the Hamiltonian is non-convex and upwind differences cannot be applied [25].

In order to solve non-convex Hamiltonian *monotone schemes* extended from hyperbolic conservation laws to H-J equations can be used [28, 31]. It was proven that monotone schemes converge to viscosity solutions and that allows to generate valid and realistic results since the entropy condition is fulfilled [31]. The key idea of these schemes is to replace the Hamiltonian H by a numerical Hamiltonian $\hat{H}$ approximation. One set of these schemes is the Lax-Friedrichs Schemes (LFS) [58]. This technique consists in replacing the Hamiltonian by the Lax-Friedrichs numerical flux function:

$$\begin{array}{c}\hat{H}(\phi) = H\left(\frac{\phi_x^+ + \phi_x^-}{2}, \frac{\phi_y^+ + \phi_y^-}{2}, \frac{\phi_z^+ + \phi_z^-}{2}\right) - \\ \alpha_x \frac{\phi_x^+ - \phi_x^-}{2} - \alpha_y \frac{\phi_y^+ - \phi_y^-}{2} - \alpha_z \frac{\phi_z^+ - \phi_z^-}{2}\end{array}, \tag{2.48}$$

where the forward and backward derivatives can be calculated using first-order approximations, such as (2.26) and (2.27) respectively, and more accurate approximations like H-J ENO or H-J WENO can also be used. Moreover, dissipation coefficients α_x, α_y and α_z have been introduced to control the amount of numerical viscosity. These factors are defined as:

$$\begin{aligned}\alpha_x &= \max\left|\frac{\partial H(\phi)}{\partial \phi_x}\right| \\ \alpha_y &= \max\left|\frac{\partial H(\phi)}{\partial \phi_y}\right| \\ \alpha_z &= \max\left|\frac{\partial H(\phi)}{\partial \phi_z}\right|\end{aligned} . \tag{2.49}$$

The straightforward method is to calculate these maximum values over the whole grid but the computational cost can be reduced by searching only in local regions close the point being evaluated. These methods are known as Local Lax-Friedrichs (LLF) [31, 52]. Furthermore, new improved versions of the LFS have been presented [59–62].

In addition, several different techniques like Godunov's scheme or the Roe-Fix scheme can be found in literature [31, 51, 63]. These methods share the same basic idea of the LFS, i.e. to use a numerical approximation of the Hamiltonian. Nevertheless, usually they require more complex implementations and a higher computational effort than LFS.

2.2.3 Dependent on local curvature

In the previous two sections, the motion of the front was defined by an external velocity field which can depend on the local geometry of the front or not. These two cases are a H-J type equation and techniques directly extended from hyperbolic conservation laws are used to define proper numerical schemes.

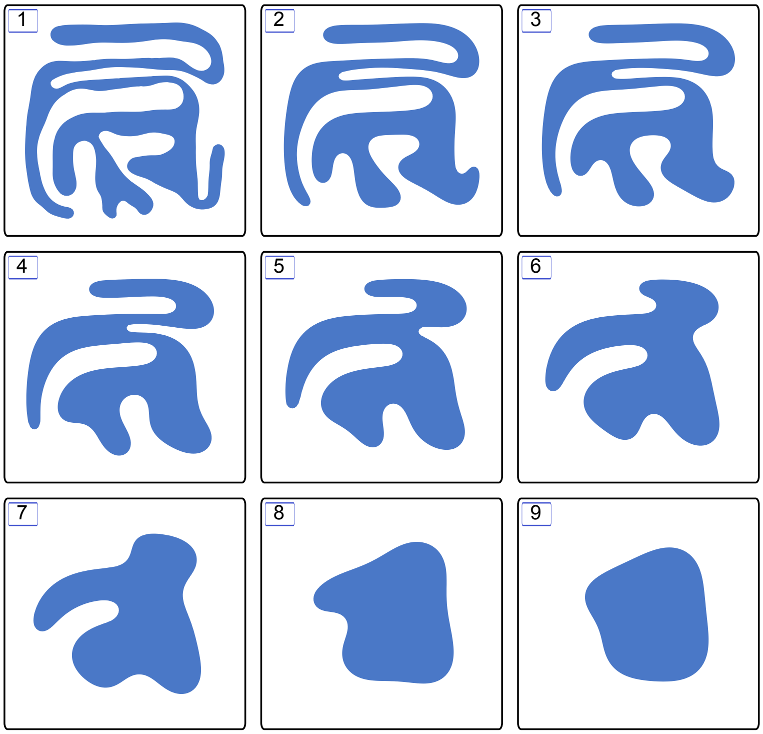


Figure 2.9: Example of a closed curve evolved by the velocity field $\vec{V} = -b\kappa\vec{N}$ with $b = 1$ in the normal direction. Reproduced from [26].

Consider now a self-generated velocity field which is proportional to its local curvature and moves the interface in its normal direction, such that

$$\vec{V} = -b\kappa\vec{N} \tag{2.50}$$

where b is a constant and κ is the local curvature. If $b > 0$ the interface moves in the direction of concavity, thus, a circumference in two dimensions shrinks to a single point and eventually disappears. Every closed curve moving with $b > 0$ will collapse to a single point regardless of its geometry complexity [64–66]. An example is reproduced in Fig. 2.9. It can be observed that the large oscillations disappears quickly while the parts with smoother curvatures remain longer. As can be deduced, the curve tends to a single point. On the other hand, if $b < 0$, the interface moves in the direction of convexity, therefore circumferences expand.

Even if the velocity field would have a tangential component V_t, this component would be automatically zero. Since $\vec{N}$ and $\nabla\phi$ point in the same direction, $\vec{T} \cdot \nabla\phi = 0$ for any tangent vector $\vec{T}$. For example, consider a velocity field

$\vec{V} = V_n\vec{N} + V_t\vec{T}$ which is used in the LS method such as:

$$\phi_t + \left(V_n\vec{N} + V_t\vec{T}\right) \cdot \nabla\phi = 0. \tag{2.51}$$

Thus, the tangential component can be avoided,

$$\phi_t + V_n\vec{N} \cdot \nabla\phi = 0. \tag{2.52}$$

Furthermore, since

$$\vec{N} \cdot \nabla\phi = \frac{\nabla\phi}{|\nabla\phi|} \cdot \nabla\phi = \frac{|\nabla\phi|^2}{|\nabla\phi|} = |\nabla\phi|, \tag{2.53}$$

equation (2.52) can be rewritten as:

$$\phi_t + V_n|\nabla\phi| = 0, \tag{2.54}$$

which is known as the *Level Set Equation* and $V_n = -b\kappa$ is the normal component of the velocity field. Finally this equation results in:

$$\phi_t - b\kappa|\nabla\phi| = 0. \tag{2.55}$$

Notice that (2.55) is a parabolic equation due to curvature dependency, thus, neither upwind differencing scheme nor LFS can be used for solving numerically this kind of motion. On the other hand, when ϕ is a SDF, the condition $|\phi| = 1$ is fulfilled and thus, the curvature can be approximated by the Laplacian of ϕ, therefore, the equality (2.36) is satisfied, leading to the heat equation:

$$\phi_t - b\Delta\phi = 0, \tag{2.56}$$

where ϕ is the temperature and b is the thermal conductivity. The heat equation is the most basic parabolic equation and these equations need to be discretized using central differencing since the information to update the front is taken from all spatial directions, as opposed to hyperbolic equations where information flows only in the direction of characteristics. To calculate κ in (2.55) second-order central differences (2.28) and (2.32) must be used in curvature (2.31) for each spatial dimension. Accordingly, the term $|\nabla\phi|$ must be calculated using second-order differences too. Likewise, the Laplacian of (2.56) is obtained with (2.37) by using second-order central differences (2.32) in each spatial dimension.

After applying a forward first-order Euler time discretization, (2.56) results in:

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} - b\Delta\phi^n = 0. \tag{2.57}$$

Nevertheless, due to the second spatial derivatives of curvature, the restrictive $O((\Delta x)^2)$ CFL condition

$$\Delta t < \left(\frac{2b}{(\Delta x)^2} + \frac{2b}{(\Delta y)^2} + \frac{2b}{(\Delta z)^2}\right)^{-1} \tag{2.58}$$

needs to be fulfilled. Analogously, (2.55) can be discretized with first-order forward Euler scheme. Notice that parabolic equations need a restrictive time step $O((\Delta x)^2)$ in contrast to hyperbolic equations such as (2.40), which need only $O(\Delta x)$. Nevertheless, enforcing $\Delta t = O((\Delta x)^2)$ gives an overall $O((\Delta x)^2)$ accurate discretization, even if forward Euler is used for time discretization.

Consider now the next *convection-diffusion equation*:

$$\phi_t + \vec{V} \cdot \nabla\phi = b\kappa|\phi|, \tag{2.59}$$

which includes both the effects of an external velocity field $\vec{V}$ and a diffusive term that depends on the curvature. Again, if the implicit function ϕ is a SDF, the equation turns into:

$$\phi_t + \vec{V} \cdot \nabla\phi = b\Delta\phi. \tag{2.60}$$

This equation can be solved applying the upwind differencing method, used in section 2.2.1 for externally generated velocity fields, to the term $\vec{V} \cdot \nabla\phi$ and using central differencing on the parabolic term $b\kappa|\phi|$ or $b\Delta\phi$. Accordingly, the most restrictive time step needs to be chosen to ensure stability, i.e. the time step imposed by the diffusive term $\Delta t = O((\Delta x)^2)$ calculated by the condition (2.58).

It is worth to pay attention to (2.60). Consider that the constant b is replaced by a term $\epsilon = O(\Delta x)$ such that the diffusion term vanishes as the mesh is refined with $\Delta x \to 0$. The equation becomes:

$$\phi_t + \vec{V} \cdot \nabla\phi = \epsilon\Delta\phi, \tag{2.61}$$

and tends to (2.38) as $\epsilon \to 0$. The addition of a diffusive term $\epsilon\Delta\phi$ to the right side of (2.38) is known as the *artificial viscosity* method. This method can be used to provide numerical stability when solving the convective term of (2.38) with central differences. The artificial viscosity method obtains the physically correct weak solution when it is taken to the limit $\epsilon \to 0$. This limit is the previously introduced *viscous limit* and corresponds to the entropy solution presented in section 2.1.2 that satisfies the Huygens' Principle. Nevertheless, when the motion is applied by the LS method, it was proven that $\epsilon\kappa|\nabla\phi|$ is a better form for the viscosity term than $\epsilon\Delta\phi$ [5, 67].

2.2.4 Local Level Set methods

All the motions previously studied require to update the implicit function ϕ over the whole grid. Additionally, it is necessary to define the velocity field $\vec{V}$ on every grid point in order to calculate the next values of ϕ when using one of the propagation equations (2.38), (2.55) or (2.56). Therefore, performing one time step over the entire computational domain requires $O(N^3)$ operations in three dimensions, where N is the number of grid points in one dimension. This method is not efficient since, usually, the application only requires one interface which is embedded in the zero level of ϕ. Thus, the rest of the ϕ levels are not essential.

The idea of local LS methods is to reduce the computational domain. This technique provides the next common advantages:

- Reduction of computational effort. Since the computational domain is reduced, the number of operations needed to updated ϕ is also reduced and is lower than $O(N^3)$.
- The extension of the velocity field over the whole grid is not always trivial since it only can make sense at the interface itself. Nevertheless, with the local LS methods, this extension is reduced and can even be avoided.
- As it is explained in previous sections, the determination of the time step Δt by CFL condition (2.45) as well as the LFS dissipation coefficients (2.49) require the search on the computational domain. Thus, since the computational domain is reduced, the number of required operations is also reduced.

2.2.4.1 Narrow Band Method

The first local LS method is known as the NBM [6, 38]. This method reduces the computational domain to only those grid points inside a narrow band. A narrow band of k layers is formed by those grid points with a distance value to the interface smaller than $k/2$ times the grid resolution. Thus, a grid point $\vec{x}_i$ will form part of the narrow band if

$$|\phi(\vec{x}_i)| \leq \frac{k}{2}\Delta x, \tag{2.62}$$

where ϕ is the SDF to the interface. For a two dimensional grid example, the ϕ values for every grid point are stored in a two-dimensional array. Additionally, the points that fulfil condition (2.62), i.e. the points inside the narrow band, are considered as *active* points and are tracked using a one-dimensional array. Thus, only the points tracked by the one-dimensional array are used for updating the values of ϕ. Likewise, the rest of grid points are kept constant. An example of an interface and the surrounding narrow band is shown in Fig. 2.10.

When the front reaches close to the edge of the narrow band, the evolution process is stopped and a new band is built such that the interface is set at the center of it. The NBM with k layers can be summarized in the simplified algorithm 2.

Since the computational domain has been reduced to only the active points inside the tube, the NBM requires only $O(kN)$ operations to update the interface in a two-dimensional space and $O(kN^2)$ in three dimensions. Although this is a significant improvement it is still not optimal and, also, additional tasks have

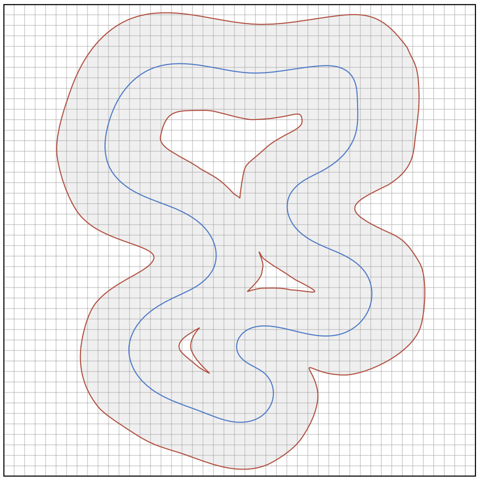


Figure 2.10: An interface (blue line) surrounded by a narrow band. Computational domain is reduced to only those gray grid points in dark area.

Algorithm 2: Simplified algorithm of the NBM

```
1 Label as *active* those points that are inside the narrow band formed by the
  points that fulfil (2.62).
2 Specify those points close to the edge of the band such that, when the interface
  reaches one of them, a new band has to be built.
3 Initialize all the points outside the band with large positive (negative) values if
  are outside (inside) the region defined by the closed interface.
4 Update the interface with the corresponding LS equation until one of the points
  defined in step 2 is reached by the interface.
5 Rebuild the band and go to 1.
```

been included, such as the definition of *active* and *edges* points, and the need to rebuild the tube every time the interface is close to the edge. Furthermore, as explained in previous sections, after updating the interface, the implicit function ϕ is no longer a SDF and, thus, reinitialization techniques must be applied.

Although the SFM, next explained, requires a lower computational cost, the NBM is still being used in such fields as: multiphase incompressible fluid simulations in combination with multi-resolution grids [68] and, also, in medical image segmentation [69]. These implementations are usually combined with techniques that maintain the LS implicit function as a SDF [70].

2.2.4.2 Sparse Field Method

The NBM is not optimal since more points than the strictly necessaries are updated. Additionally, the narrow band has to be defined periodically as the interface is evolved and a reinitialization technique must be applied to keep the implicit function as a SDF.

Consequently, the SFM was introduce by Whitaker [7]. The SFM only updates those points strictly necessary to calculate the next step of the interface. In the SFM, several lists are used for keeping track of the grid points that are required to update ϕ with the LS equation. The number k of required lists depends on the adjacent points used for computing the new values of ϕ at interface points. Therefore, if n forward points and n additional backward points are used, the total number of lists that has to be updated is $k = 2n + 1$, such that $L_{-n}, L_{-n+1}, \cdots, L_{+n}$ are all the lists. Notice that the list containing the interface grid points, namely L_0, is also included. Each of these lists, L_i, contains a layer of points that fulfil condition:

$$i - \frac{\Delta x}{2} \leq \phi(\vec{x}) \leq i + \frac{\Delta x}{2}. \tag{2.63}$$

Similarly to the NBM, in the SFM the active grid points act like a border between the positive and negative regions, while the rest of the grid points are kept constant. An example of the computed grid points by both methods is shown in Fig. 2.11. In the SFM example, only the interface points and their adjacent ones are updated, i.e. $k = 3$, whereas in the NBM about $k = 11$ layers of points are included in the computational domain.

Once the list of points are defined, the ϕ values of the list L_0 are updated with the LS equation and, depending on the new distance value of each point, it is moved to the corresponding adjacent list, i.e. $L_{\pm 1}$. Then, the points of the rest of the lists are updated by simply adding/subtracting Δx (assuming a regular LS grid) to its closest point of the adjacent list depending if they are positive/negative. For instance, the points of the list L_{+1}, are updated by adding Δx to the closest points

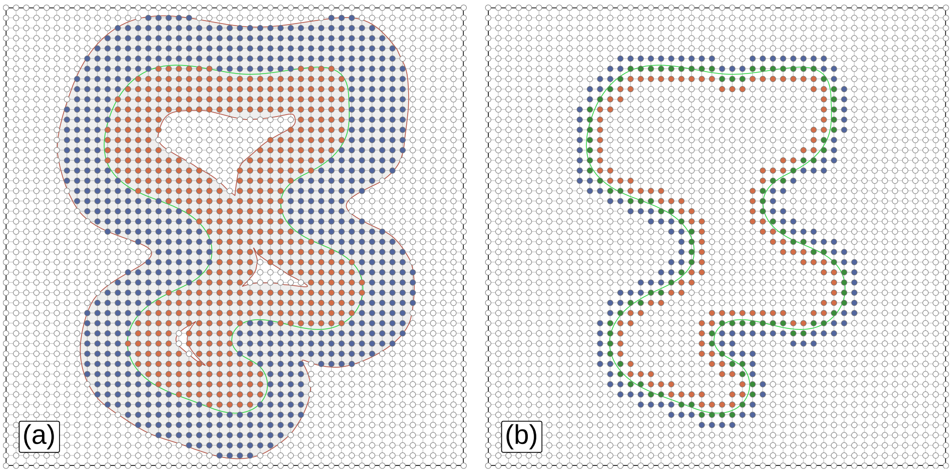


Figure 2.11: Comparison of the grid points included in the computational domain in: (a) the NBM and (b) the SFM. The interface is represented by a green line and the noncomputed points are shown in white. In (a) the blue/orange points are the exterior/interior computed points. Whereas in (b) the green, blue and orange points are those included in L_0, L_{+1}, L_{-1} SFM lists.

of L_0, and the points of L_{-2}, update their values subtracting Δx to the closest points of L_{-1}. After the distance values are properly updated, those points next to an interface point that do not belong to any list, are included in the corresponding most exterior list depending on their signs. Thus, a continuous process is enabled such that, as the interface evolves, new grid points are included in one of the most exterior list $L_{\pm n}$ while other points are transferred to the adjacent lists and, when they are far enough from the interface, they do not belong any longer to any list and are kept constant.

In order to ease the calculations of this process, every grid point is labelled with a *state* according to the list it belongs to, such that a point belonging to L_i is labelled with state i. Moreover, if a negative point does not belong to any list it is labelled with state $-(n+1)$, where n is the subscript of the most exterior list. Analogously, an exterior point that do not belong to any list is labelled with state $(n+1)$. Also, these points are kept to the constant values $\frac{k\Delta x}{2}$ and $-\frac{k\Delta x}{2}$, respectively, where k is the number of total layers of points.

Additionally, in order to do not change the state of the points and their associated lists until all of the points are updated, auxiliary lists are defined to temporary store the information of the points that have to be transferred to another list. Therefore, in a SFM with k lists, the auxiliary lists are defined as

$S_{-n}, S_{-n+1}, \cdots, S_{+n}$, such that the list S_i contains the information about those points that will be transferred to the list L_i at the end of the evolution process.

As an example, consider a simple scenario where first-order differences (forward and backward) (2.26) and (2.27) are used. Thus, only $k = 3$ layers of grid points are necessary to update ϕ, namely: L_0 includes those points that contain the interface itself, layer L_{+1} contains the positive points adjacent to the interface, and those negative points adjacent to the interface are included in L_{-1}. Moreover, the auxiliary lists S_0, S_{+1} and S_{-1} are also defined. The grid points are classified depending on its distance to the interface as compiled in table 2.1.

Set of points	state label	ϕ interval value
L_0	0	$[-0.5\Delta x, 0.5\Delta x]$
L_{+1}	1	$(0.5\Delta x, 1.5\Delta x]$
L_{-1}	-1	$[-1.5\Delta x, -0.5\Delta x)$
Rest of interior	-2	< -1.5
Rest of exterior	2	> 1.5

Table 2.1: Corresponding SFM states and lists of grid points depending on their distance values.

Because of simplicity, the algorithm of the SFM is explained according to the first-order differences example for a three-dimensional case. Consider the interface is already defined, thus, in order to evolve it according to the SFM, the algorithm formed by the consecutive evaluation of procedures 3, 4, 5, 6, and 7 must be applied.

The SFM is based on the approximation that the points next to the interface evolve exactly like the interface itself. Therefore, there is no need to apply any velocity extension technique. Furthermore, since the neighbouring points of the interface are updated with perfect distance values, a SDF is built in each time step and, thus, no reinitialization process is required.

The accuracy of the SFM was compared to the traditional LS method [7]. In order to measure the error of both approximations, the evolution of a circle was simulated by both techniques since it can be calculated analytically and, thus, used as reference. To compute the total error of a method, first the zero-contour of the implicit function ϕ was obtained and then, the distance to the analytical solution was computed. To find the points that form the contour, a linear interpolation between adjacent points that lie on either side of a zero level was used. Then, the exact point where these lines have a zero value was used for calculating the distance to the analytical circle. Finally, the total error is the root mean squared of

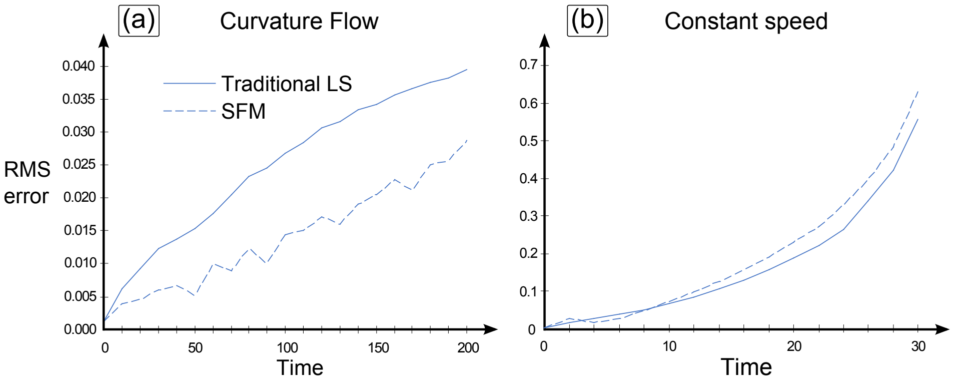


Figure 2.12: Error comparison of the traditional LS method and the SFM in two different scenarios: (a) a circle moving under its own curvature and (b) a circle moving in the direction of the inward normal with constant speed. Figure adapted by author from [7].

these distances. The results of this study of the error are reproduced in Fig. 2.12. Two examples were studied, both consisting of a moving circle but the first one was evolved under its own curvature 2.12(a) whereas the second one 2.12(b) was moved in the direction of the inward normal at a uniform speed of 1. The results show that both methods produce comparable errors and are within the same order of magnitude although the SFM can obtain lower errors than traditional LS method.

Procedure 3: Initialization

```
1 Determine the SDF $\phi$ over the grid. Notice that this calculations can be limited
  to only those active points required by the SFM.
2 Build the lists and label the points according to table 2.1.
```

Procedure 4: Evolution

```
1 for each $L_0$ point $\vec{x}_i$ do
2     Update $\phi(\vec{x}_i)$ value according to the LS equation.
3     if $\phi(\vec{x}_i) > 0.5\Delta x$ then add $\vec{x}_i$ to $S_{+1}$
4     if $\phi(\vec{x}_i) < -0.5\Delta x$ then add $\vec{x}_i$ to $S_{-1}$
```

Procedure 5: Lists update

```
1  for each $L_{+1}$ point $\vec{x}_i$ do
2      Among the six neighbouring points of $\vec{x}_i$, find the point $\vec{x}_b$ with the minimal $\phi$ value and $state = 0$.
       if no point with $state = 0$ is found then
3          remove $\vec{x}_i$ from $L_{+1}$
4          $state(\vec{x}_i) = 2$
       else
5          Do $\phi(\vec{x}_i) = \phi(\vec{x}_b) + \Delta x$
6          if $\phi(\vec{x}_i) \in [-0.5\Delta x, 0.5\Delta x]$ then add $\vec{x}_i$ to $S_0$
7          if $\phi(\vec{x}_i) > 1.5\Delta x$ then remove $\vec{x}_i$ from $L_{+1}$
8  for each $L_{-1}$ point $\vec{x}_i$ do
9      Among the six neighbouring points of $\vec{x}_i$, find the point $\vec{x}_b$ with the maximal $\phi$ value and $state = 0$.
       if no point with $state = 0$ is found then
10         remove $\vec{x}_i$ from $L_{-1}$
11         $state(\vec{x}_i) = -2$
       else
12         Do $\phi(\vec{x}_i) = \phi(\vec{x}_b) - \Delta x$
13         if $\phi(\vec{x}_i) \in [-0.5\Delta x, 0.5\Delta x]$ then add $\vec{x}_i$ to $S_0$
14         if $\phi(\vec{x}_i) < -1.5\Delta x$ then remove $\vec{x}_i$ from $L_{-1}$
```

Procedure 6: Transfer from auxiliary lists

```
1  for each $S_0$ point $\vec{x}_i$ do
2      remove $\vec{x}_i$ from $S_0$
3      add $\vec{x}_i$ to $L_0$
4      $state(\vec{x}_i) = 0$
5  for each $S_{+1}$ point $\vec{x}_i$ do
6      remove $\vec{x}_i$ from $S_{+1}$
7      add $\vec{x}_i$ to $L_{+1}$
8      $state(\vec{x}_i) = 1$
9  for each $S_{-1}$ point $\vec{x}_i$ do
10     remove $\vec{x}_i$ from $S_{-1}$
11     add $\vec{x}_i$ to $L_{-1}$
12     $state(\vec{x}_i) = -1$
```

Procedure 7: Evolution of exterior lists

```
1 for each $L_0$ point $\vec{x}_i$ do
      for neighbouring points $\vec{x}_b$ with state $= 2$ do
2         add $\vec{x}_b$ to $L_{+1}$
3         $state(\vec{x}_b) = 1$
      for neighbouring points $\vec{x}_b$ with state $= -2$ do
4         add $\vec{x}_b$ to $L_{-1}$
5         $state(\vec{x}_b) = -1$
6 Go to step 1 of procedure 4 until evolution is performed.
```

After the presentation of the SFM, variations of this method have been presented. For example, in the field of medical image segmentation, an SFM that approximates the implicit function ϕ by only three integer values, namely, -1, 0 and $+1$ for interior, interface and exterior points was published [71]. Also further improvements were presented [72, 73]. These methods speed up significantly the simulations since only basic calculations are performed. Nevertheless, the accuracy of the results is reduced in comparison with original SFM.

2.2.4.3 Further optimizations

Despite the SFM reduces significantly the computational cost of the LS method from $O(N^3)$ to $O(N^2)$ in three-dimensional scenarios, $O(N^3)$ memory space is still required since the ϕ values of every grid point must be stored.

This problem can be relieved using *octrees* data structures, which are very efficient in terms of storage since only $O(N^2)$ space is required [74] due to the mesh refinement, i.e. only a few grid points are used far from the interface but a finer one is used close to it. However, the data access time is increased from $O(1)$ up to $O(\log N)$ because of the hierarchical storage. Several LS implementations can be found which use octree techniques [75–78].

Another data structure that has been used to reduce storage requirements is the Run-Length Encoding (RLE) LS method [79]. This technique applies the RLE scheme to compress regions far from the interface to only their sign representation while storing with full precision the active points. The storage efficiency is further improved over the octree LS.

Furthermore, a technique that keeps the optimal time access $O(1)$ and improves the space storage requirements was presented [80]. This method, known as *sparse block grid*, divides the three-dimensional N^3 space into small cubic blocks of m^3 points each. Then, the coarse grid of $(N/m)^3$ stores pointers to only those cubic blocks that intersect the interface. These blocks are constantly allocated and deallocated as the interface is evolved. This method provides a storage complexity of $O\left((N \cdot m)3 + m^3 n^2\right)$.

Recently, the *Hash Table Local* LS method was introduced [81]. This method only computes the LS data in a band around the interface, like the SFM but, additionally, only stores the data in that same band. Then, a hash table data structure is used for accessing stored data. This kind of data structure provides an $O(1)$ access time. Nevertheless, the authors conclude that their method, while being easier to implement, performs worse than a quadtree implementation.

2.3 Graphics Processing Units

A Graphics Processing Unit (GPU) is a specialized electronic circuit designed to calculate arithmetic operations related with computer graphics visualization or rendering. Nowadays, GPUs are included in embedded systems, mobile phones, personal computers, workstations, and game consoles. Their first goal is to reduce computational effort to the Central Processing Unit (CPU) in applications, such as video games or interactive three-dimensional applications, by manipulating computer graphics and performing image processing. Meanwhile, the CPU can be used exclusively for other algorithms like mathematical operations.

GPUs are usually designed for applications that requires a large computational effort such as real-time rendering in high resolution three-dimensional video games. Moreover, these applications also require many operations per pixel, for instance, in order to apply anti-aliasing filters to smooth the scene and to provide a more realistic aspect.

The input of a GPU is a list of geometric *primitives*. These primitives are the simplest geometric objects that the system can handle, typically triangles in a three-dimensional application. Many operations are applied to these primitives, such as shading, in order to map them onto the screen and to create a final picture. Historically, some of these operations were configurable but not programmable. Next generations of GPUs could implement many different operations to the geometric primitives and, eventually, these operations became completely programmable.

Consequently, over the past years, the scientific community has identified other applications unrelated with graphical operations but with similar characteristics. Therefore, due to the high programmability gained during last years, the GPUs were used as *general-purpose* computing units. Additional reasons of the quick expansion of the General-Purpose computing on Graphics Processing Unit (GPGPU) are [82]:

- The great relation computing-power/price.
- The huge integration on personal computers since the most of them have the capability of executing and developing GPGPU algorithms [83].
- High-level programming languages and tools created by GPU vendors that allow programmer flexibility, productivity and an easy learning process.
- Communities created by vendors in order to make public their own GPUs and their capabilities. Events, awards, fellowships and seminars are financed by the biggest vendors (like Nvidia) every year.

GPUs were traditionally focused on graphic operations and many of these operations can be calculated for each pixel independently of the rest of the pixels. Therefore, parallelism is a natural way of increasing performance in GPUs. High computational requirements of graphic operations, especially in video games, caused a drastic improvement of the computational capabilities of GPUs. Eventually, they became a highly parallel programmable processing unit and started forming part of the *many-core* family processors since the execution of a lot of simultaneous threads is maximized. A many-core processor example is the Nvidia GeForce GTX Titan Z GPU which has 5760 cores [84].

GPUs are not the only many-core architecture that has gained interest over the last years. Field-Programmable Gate Arrays (FPGAs) have been highly successful in many application areas [85]. Although in some applications FPGAs can obtain better performances [86], GPUs are still the most used many-core platform due to the previously cited reasons and, also, they provide faster time to deployment than FPGAs [87]. This is because FPGAs are, unfortunately, hard to program for general-purpose computing since it requires detailed knowledge of low-level hardware. Nevertheless, the FPGA vendor Altera has recently announced support for a high-level programming language to implement parallel environments. Therefore, the FPGAs have nowadays become an attractive many-core choice and several applications have been implemented recently and compared with GPU implementations [88]. Generally, better energy efficiency is shown by the FPGAs implementations and in, some cases, they are even faster than GPUs [88, 89].

On the other hand, the other family processors that exists nowadays is the *multi-core* processors like modern CPUs. Current CPUs, rather than increasing the features of single core, are integrating several cores. Multi-core devices maximize the performance of running sequential instructions, enabling only a few simultaneous execution threads. Traditionally, one of the major performance factors of CPUs has been increasing frequency. However this increase was stopped at just below 4.0 GHz due to the required huge power consumption [85, 90]. Consequently, CPUs started increasing performance through multi-core and vector

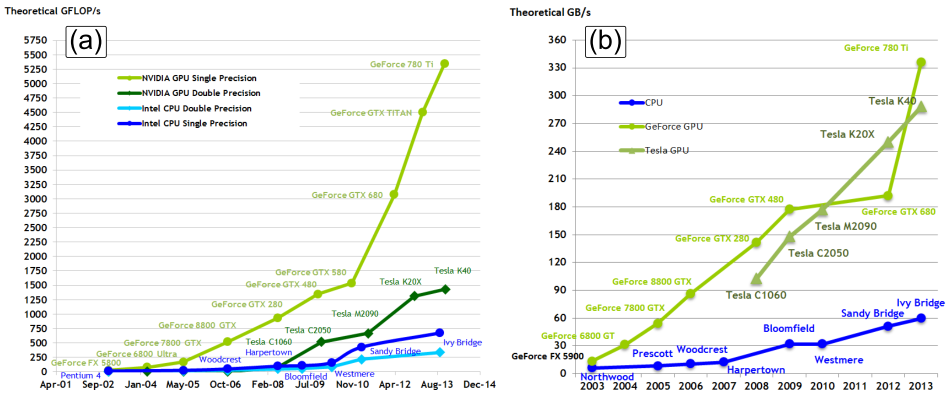


Figure 2.13: Historical comparison of theoretical peak performance in terms of: (a) gigaflops and (b) bandwidth for the fastest available Nvidia GPUs and Intel CPUs. Reproduced from [92].

instructions instead. An example of this kind of processors are the Intel *Xeon E7* that currently can integrate up to 15 cores, 2 execution threads per core and it can operate up to 3.4 GHz [91].

In Fig. 2.13 a historical and theoretical performance comparison of the fastest Nvidia GPUs and Intel CPUs in terms of gigaflops and bandwidth is presented. Data from last year shows that GPU performance of both metrics is roughly between 4 and 8 times higher. Accordingly, a massively parallelized algorithm executed on a GPU can be up to seven times faster than a parallel CPU implementation [20].

At present, legacy systems that have been used for several years need to evolve in order to take advantage of new multi-core or many-core processors [93]. Nevertheless, the increasing of parallelism only improve the performance of those parts of the algorithm suitable to be parallelized, meaning that the serial section of the code can become the bottleneck [94]. Thus, the majority of the applications benefit from the combination of both a many-core GPU and a fast multi-core CPU.

2.3.1 Graphics pipeline GPUs evolution

Before GPUs were used for GPGPU, they were used exclusively for graphics operations in order to render images with some visual effects. This process can be understood as set of stages that are applied sequentially and it is known as the *graphics pipeline*. A GPU is a hardware implementation of the pipeline and an Application Programming Interface (API) is used for rendering the desired graphics. An API provides a standardized way for programmers to develop graphics rendering capable of being executed in any hardware that supports the specific API. Then, the set of instructions generated by the API are interpreted by the drivers of the GPU.

Following, the different stages of a graphics pipeline of an early Nvidia GeGorfe are commented [95]:

- Interface. The link between the CPU and the GPU. Through this interface both, API instructions and data, are received.
- Vertex control. GeForce GPUs only interpret triangle primitives from CPU. This stage, then, converts the triangle data into a form that the hardware understands and places the prepared data into the vertex cache.
- Vertex shading, transform and lighting (VS/T&L). The primitives are modified by applying some transform operations to their vertices, such as rotation or translation. Additionally, some values like colors, normal vectors, and tangents, are calculated.
- Triangle setup. Edges equations are created to interpolate colors and other vertex properties.
- Raster stage. It determines which pixels are contained in each triangle. Then, for every pixel, some necessary values for shading it are interpolated.
- Shader. The final color of each pixel is determined.
- Raster operation stage. It performs the final raster operations on the pixels. Transparency and antialiasing effects are computed. It also determines the visible/occluded objects according to the given view point.
- Frame buffer interface. This stage manages memory reads/writes from/to the display frame buffer memory that is used for finally representing the image on the display.

This pipeline was upgraded over the years, improving rendering of graphics. Originally, the vertex and pixel shader stages were configurable but not programmable. Nevertheless, the version 8 of the API DirectX included a programming language similar to assembly with a set of 127 instructions. Thus, developers could use them on the supported GPUs in order to apply different graphic operations. The first GPU that supported these instructions was the Nvidia GeForce 3 and it appeared in the 2001.

GPU architectures have focused increasingly on the programmable parts of the graphics pipeline. Indeed, previous generations of GPUs could be understood as additional instructions to a fixed-function pipeline. However, current GPUs can be defined as a programmable device surrounded by supporting fixed-function units [82]. In 2006 the GeForce 8800 GPU was introduced. This device mapped the separate programmable graphics stages into an array of unified processors, such that the logical graphics pipeline is a loop with much fixed-function graphic logic between the iterations. This unification enables a better load balance by dynamically allocating the execution resources to different pipeline stages.

The GeForce 8800 was served with the DirectX 10 API generation. By this generation, the vertex and pixel shader stages had been unified to a new logical stage, namely geometry shader. The geometry shader stage processes all the vertices of a primitive rather than isolated vertices, such that both stages are identical to the programmer. Additionally, innovative features were introduced such as: 32-bit integer and floating-point support, arbitrary number of reads from global memory, and dynamic flow control set instructions.

GPUs continued evolving and including new functionalities until current GPUs, which are formed by an array of thousands of processing units. These processors are completely programmable, are able to execute code simultaneously and support 64-bit integer and floating-point instructions. This enables the possibility of using GPUs as massive parallel computing devices in many scientific applications, such as: simulation of information propagation over networks [96], acceleration of fluids dynamics [97], simulation of stars clusters around black holes [98], medical image processing [99–101] and micromachining processes simulation [19, 20, 102].

2.3.2 GPGPU programming languages

The first time that GPGPU was considered of interest was in 2003 [103]. Originally, fixed graphic-functions (e.g. texture filters and blending) were used in order to perform GPGPU operations. This was extremely improved by the introduction of high-level programming languages that were designed for computation and abstracted of the API graphics of the GPU. Two of the first ones were BrookGPU [104] and Sh [105] (which later was commercialized as RapidMind [106]), both were academic research projects with the goal of abstracting the GPU as a streaming processor.

Consequently, the majors GPU vendors made public their own languages in order to extend the usage of their devices as research instruments. There are three major GPU vendors for the PC market, Intel being the largest. Nevertheless, Intel is only dominant in integrated and low-performance market. On the other hand, Nvidia and AMD are the two unique suppliers for high-performance GPUs. AMD released their own language to researchers in 2006, namely Close To Metal (CTM). CTM was a low-level programming interface but was short-lived since AMD subsequently switched from CTM to Open Computing Language (openCL) [107].

OpenCL is a framework, very similar to C programming language, for writing programs that execute across heterogeneous platforms such as CPUs, GPUs, digital signal processors, FPGAs and other processors[2]. The openCL project was initially developed by Apple Inc. and they submitted their initial proposal to Khronos Group but, eventually, was supported by AMD, IBM, Qualcomm, Intel and Nvidia. The first version of openCL was released on December 2008. This framework has severely extended due to all the support that is receiving from important companies, like Altera that has recently presented an openCL compiler to be directly used in their FPGAs devices [109].

On the other hand, the Compute Unified Device Architecture (CUDA) platform presented by Nvidia has been very successful since its release in June 2007. Despite of Nvidia GPUs also support openCL, CUDA is the most mature technology with the most advanced development tools. The CUDA platform is accessible to developers through CUDA-accelerated libraries, compiler directives (such as OpenACC), and extensions to standard programming languages, including C, C++, Fortran and Python among others.

Another key aspect for the quick expansion of the GPGPU environments like CUDA and openCL has been the Software Development Kits (SDKs) presented by the vendors. For example, the AMD APP SDK or the CUDA toolkit that includes a complete integration in Microsoft Visual Studio in Windows systems or a customized version of Eclipse.

Despite of the variety of languages, all of them are conceptually equivalent and offer roughly the same functionalities. Nevertheless, in the present thesis, the CUDA platform has been chosen due to the large amount of information including books, tutorials, templates and the high popularity that has achieved in last years.

2.3.3 CUDA platform

In this section the architecture of modern Nvidia GPUs, i.e. CUDA, is described. Nevertheless, modern GPUs of other vendors like AMD implement very similar structures as explained in section 2.3.1. Furthermore, notice that several CUDA microarchitectures have been presented since its release including significant improvements.

The Nvidia GeForce 8800 GPUs in 2007 were the first devices implementing CUDA. These devices implement the first CUDA microarchitecture, namely Tesla[3]. In the present thesis, an Nvidia GeForce GTX 260 which implements

[2]Heterogeneous computing refers to systems that use more than one kind of processor to gain performance. For example, a system formed by a CPU and a GPU [108].

[3]Do not confuse the Tesla microarchitecture with the Tesla GPUs specially designed for servers and workstations.

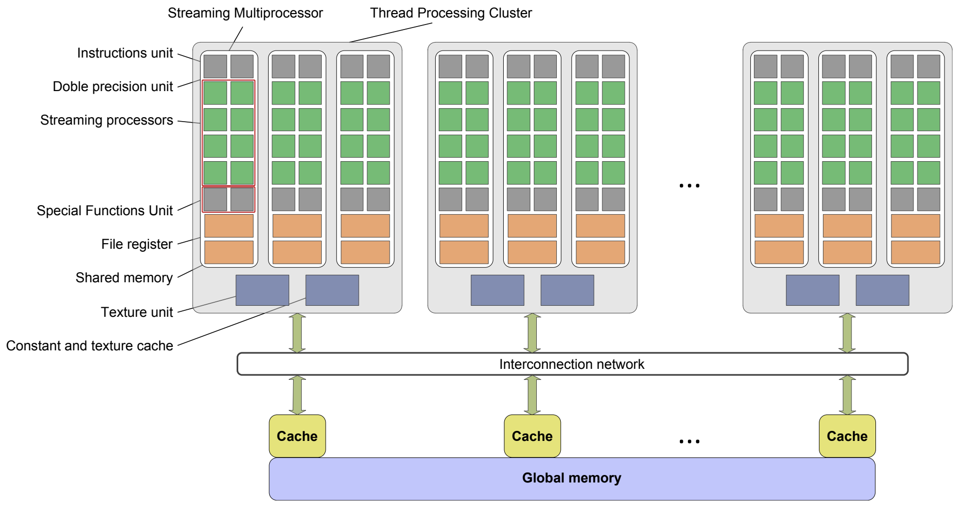


Figure 2.14: Tesla CUDA microarchitecture schematic of the Nvidia GT200 series [110].

the Tesla microarchitecture but with some improvement with respect to GeForce 8800 series, was used, thus, the GT 200 series is explained.

Tesla CUDA microarchitecture is the first one implementing the unified shader model and is composed of hundreds of simple cores that are designed to maximize floating-point throughput. Each CUDA core has a fully pipelined integer Arithmetic Logic Unit (ALU) and a Floating-Point Unit (FPU) that executes one 32-bit integer or floating point instruction per clock cycle. The CUDA cores are grouped into Streaming Multiprocessors (SM), each with 8 CUDA cores, also including 2 special functions units and one instructions unit that control the execution of the threads on the multiple processors as well as a double precision 64-bit processing core. Moreover, a 32-bit register file, a data parallel shared memory, a cache for constant data, and another reading texture cache are included in the SM. Likewise, the SMs are organized in Thread Processing Clusters (TPCs) such that SMs can share texture, cache and constant memories. Finally, the TPCs are connected to the global memory. A schematic of the described architecture is depicted in Fig. 2.14.

CUDA is designed to execute a lot of threads simultaneously. For instance, the GPU Nvidia GeForce GTX 285 has 240 CUDA cores. The Tesla generation contains 10 TPCs of 3 SMs each, resulting in 30 SMs per GPU. Moreover, each SM is able to handle up to 1024 simultaneous threads, resulting in 30, 720 threads per GPU [110].

2.3.3.1 Parallel execution

A subroutine executed in a CUDA GPU is called *kernel* and it is invoked by the CPU. When an application invokes a kernel, the organization of execution threads must be defined. CUDA platform uses a hierarchical organization of threads such that, the threads are grouped into *blocks* and the blocks are distributed over a *grid*. Both, the size of the thread blocks and the dimensions of the grid, must be provided when invoking a kernel and this will determine the number of total execution threads. This enables the possibility of efficiently adapt the structure of threads to the hardware while executing an algorithm in a natural an optimal way.

Maximum block and grid dimensions are determined by CUDA microarchitecture. For instance, Tesla microarchitecture supports thread blocks up to 512 threads that can be distributed in 3 dimensions whereas a maximum number of 65535 blocks distributed only in two dimensions is supported.

When a kernel is invoked, each SM receives a block of threads and the blocks are computed simultaneously. Once a SM has finished computing its corresponding block, its resources are released and a new block is delivered to the now free SM. This enables a workflow such that the blocks wait to be computed until a SM is free. Once every block has been computed, the kernel execution is over. A simplified kernel execution of a sum operation of two arrays is shown in Fig. 2.15. Initial arrays $v1$ and $v2$ have 1024 positions each, thus, the natural way of performing a sum operation is creating 1024 threads so that each thread only performs the operation $v1[id] + v2[id]$, where id is the thread identity $id = 0, 1, 2, 3, \cdots, 1023$. Additionally, this result is stored in the result variable r, such that $r[id] = v1[id] + v2[id]$. Accordingly, one way to group 1024 threads is creating 128 blocks of 8 threads each. After the creation of the blocks, the first 30 are assigned to the 30 SMs that has de GPU while the rest of the blocks wait until a SM finishes computing the current block and its resources are released. This process is continued until every block has been computed.

Notice that this is just a simple example and probably is not the optimal choice. Best choices are those that the size of the block is multiple of 32 such as 128 or 256 [92]. This is because the threads within a block are grouped into *warps*, each containing 32 threads. This hierarchical organization is depicted in Fig. 2.16. All the threads of a warp execute the same code and can be alternated with other warps within the same SM. A SM can manage up to 32 warps simultaneously in the Tesla microarchitecture although it only contains 8 cores. In order to accomplish managing such a number of warps with those limited resources, the delays produced when a warp accesses to the global memory are used for other warps to perform operations. Typically, the delays produced by the global memory, which is off-chip, are of the order of several hundreds of clock cycles [111] thus,

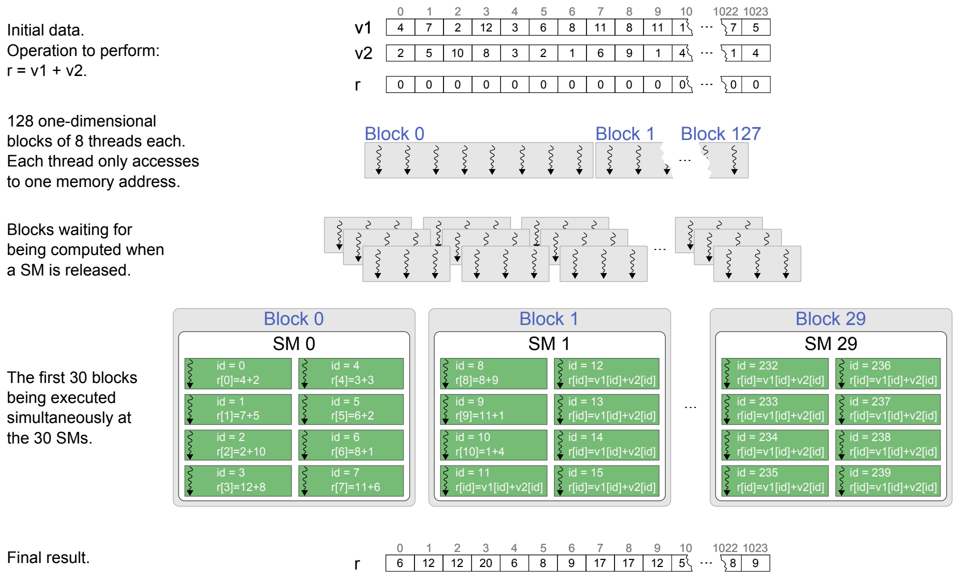


Figure 2.15: Workflow of a kernel execution. The kernel consists in performing the operation $r = v1 + v2$.

other warps can use the now unused core. The swapping warps process is depicted in Fig. 2.17.

A GPU can be used as a massively parallel computing unit. Nevertheless, in order to take completely advantage of the GPU power, the application being executed must be capable of being split in many execution threads which apply the same operations to different elements. In addition, the number of threads must be high enough to use all the cores of the GPU. Otherwise the acceleration provided by the GPU with respect to a traditional sequential execution can be insignificant or even a worse computational efficiency can be obtained.

2.3.3.2 Memory structure

Another important aspect to consider when developing a CUDA algorithm is the communication between execution threads. As has been explained previously, accessing to the global memory requires many clock cycles thus resulting inefficient. CUDA platform implements a hierarchical structure of memories to provide an efficient communication between threads within the same block. Following, the Nvidia CUDA hierarchical memories are commented:

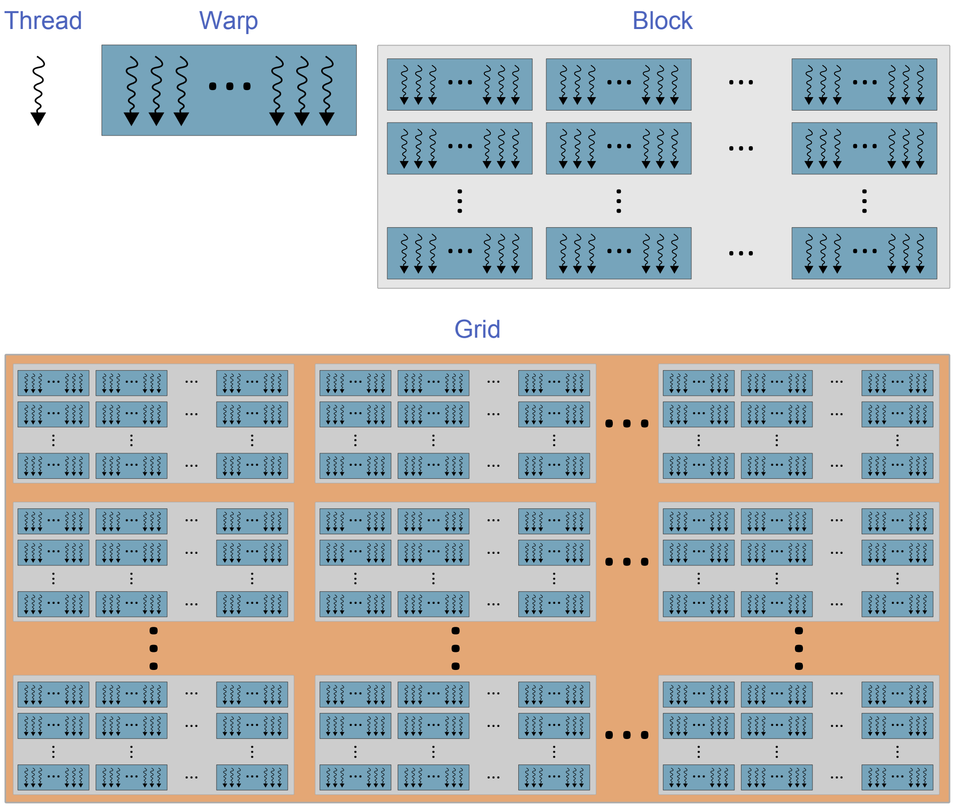


Figure 2.16: CUDA hierarchical organization of execution threads.

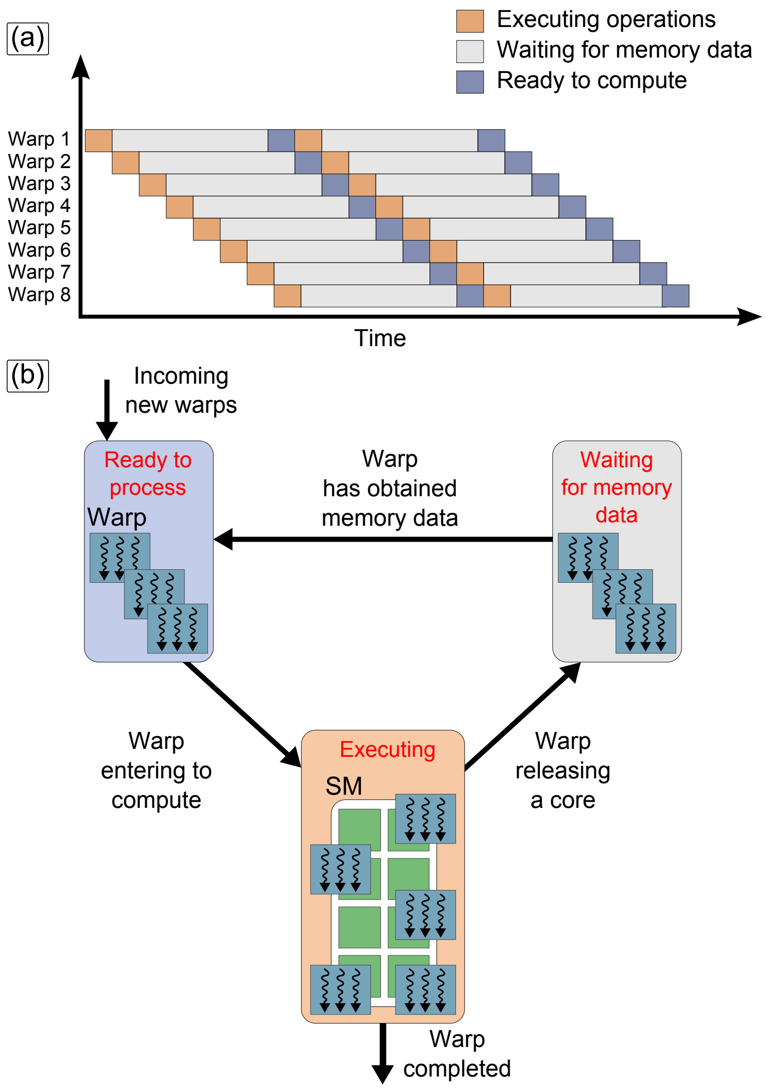


Figure 2.17: Warp swapping process. This figure shows how the delays produced when a warp accesses to the global memory are taken in advantage by other warps. (a) shows the different states of a warp in time whereas in (b) the flow chart of execution is shown.

- *Per-thread registers.* There are 16 K 32-bit registers per SM and they store the local variables of each thread, i.e. only one thread can access to its own registers. Notice that these registers are shared between all the concurrent threads within a block, thus, the higher the number of threads per block, the lower the number of registers that can be used by one single thread.

- *Shared memory.* 16 KB of memory included in each SM. The data stored in the shared memory can be accessed by all the threads within a block but it will be deleted when the execution block finishes. To achieve high bandwidth, shared memory is divided into 16 equally-sized memory banks, which can be accessed simultaneously. However, if two addresses of a memory request fall in the same memory bank, the access is serialized. Thus, special attention to the memory accesses has to be paid in order to maximize the bandwidth. Since it is an on-chip memory only a few clock cycles are necessary to access [92].

- *Global memory.* This is the main memory of the device and can vary from hundreds of MB to several GB of capacity. Every execution thread has access to the whole memory. Since it is an off-chip memory, high latencies are produced when accessing to it. Allocation and releasing memory are performed from the CPU and the data stored is maintained during the whole application. In the first GPUs, a *coalesced access* had to be performed in order to get an optimal bandwidth data transfer [92].

- *Constant memory.* The CPU can store variables in global memory as constant data which is cached and cannot be modified by GPU.

- *Texture memory.* The texture memory space resides in global memory and is cached in texture cache. Then, if the requested data is stored in the cache, the access time is reduced. This cache is optimized for 2-dimensional spatial locality so, threads of the same warp that read close together texture addresses will achieve the best performance. Thus, coalesced accesses are easily obtained, reducing reading time.

2.3.3.3 Kepler microarchitecture

During the last part of the thesis, an Nvidia GeForce GTX Titan was used for performing calculations. This GPU implements the Kepler microarchitecture, therefore, the main differences with respect to the Tesla microarchitecture are following commented.

In previous microarchitectures (Tesla and Fermi) the 2x shader clock was used thus, by running units at higher clock rate, a higher throughput can be achieved with fewer execution units. This enables an area optimization but it consumes more power.

GPU features	**Tesla**	**Kepler**
Maximum number of SM	30	15
Max. CUDA cores	240	2880
Max. threads per block	512	1024
Max. blocks in a grid	65535	$2^{31}-1$
Max. dimensions of the grid	2	3

SM features		
Max. cores	8	192
Max. threads	1024	2048
Special functions units	2	32
Instruction dispatch units	1	8
Double-precision cores	1	64
Number of 32-bit register	16 K	64 K
Shared memory	16 KB	48 KB
Shared memory banks	16	32

Table 2.2: Main differences between Tesla and Kepler CUDA microarchitectures.

Kepler architecture is the first focused on efficiency, programmability and performance [112]. The efficiency goal was achieved through the use of a unified clock instead of the $2x$ shader clock while including more cores to achieve similar levels of performance. The reduction of clock rate reduces power consumption down to 50% and, additionally, two Kepler cores use about 90% of the power of one of the previous Fermi architecture.

For example, the Nvidia GeForce GTX 560 is one of the last GPUs implementing the Fermi architecture. This device has 336 CUDA cores running at 1700 MHz [113]. On the other hand, the Nvidia GeForce GTX 660 is one of the first GPUs that implements the Kepler microarchitecture. This GPU has 960 CUDA cores but running only at 980 MHz due to the new efficiency goal of this architecture [114].

As a consequence of the above, Kepler architecture employs a new SM architecture called SMX which includes much more CUDA cores. This allows to execute a whole warp per clock cycle. A SMX includes 192 CUDA cores.

Another difference is that Tesla GPUs use IEEE 754-1985 floating point arithmetic, however, Kepler architecture implements the new IEEE 754-2008 floating-point standard. This provides the fused multiply-add (FMA) instruction for both single and double precision arithmetic. The FMA performs multiplication and addition with a single final rounding step unlike a multiply-add instruction. Thus, no precision is lost in the addition and the FMA is more accurate than performing the operations separately [112].

Finally, differences of the features between Tesla and Kepler microarchitectures are grouped in Table 2.2 [92, 110, 112].

CUDA platform has been extensively used in many research fields such as computational finance [115, 116], astronomy related [98], medical image [99, 100], simulation of clustering process of protein structures [117], parallel data mining [118] and defence-related applications [119, 120]. Particularly in this thesis, the power of Nvidia GPUs has been used for accelerating the simulation of wet and dry etching processes applied to micromachining of microstructures [19, 20, 102].

2.4 Micro-Electro-Mechanical Systems (MEMS)

The main topic of the thesis is the improvement of micromachining processes simulation by means of the LS method. The application of these processes is the fabrication of microstructures based, mainly, on silicon (Si) or quartz substrates. Therefore, in this section an introduction to such structures as well as the corresponding fabrication methods are given.

2.4.1 Introduction

The acronym Micro-Electro-Mechanical Systems (MEMS) was used in the late 80's but the first fabrication process of a device that can be considered a MEMS was patented in the 50's [121]. Originally, the MEMS technology was used for referencing to electrical and mechanical devices within microscopic scale. These devices are three-dimensional structures with a specific electrical and/or mechanical behaviour. However, in the last twenty years, the MEMS field has become a high-technology scientific field that includes many different kinds of devices with varied applications. A device can be currently considered a MEMS if characteristic sizes of its structures are within the $0.1 - 1000$ μm range in either of the x, y or z dimension. In addition, a MEMS should include at least one of the following features: multiple components, complex functions, system integration and the capability of mass production [122]. Nowadays it is possible to fabricate devices smaller than $0.1\mu m$, however those devices are included in the Nanotechnology field and, then, the new acronym Nano-Electro-Mechanical Systems (NEMS) is used.

Nowadays, MEMS technology can be found in many applications such as micromirrors and microlens[123], high-quality tunable Radio Frequency (RF)

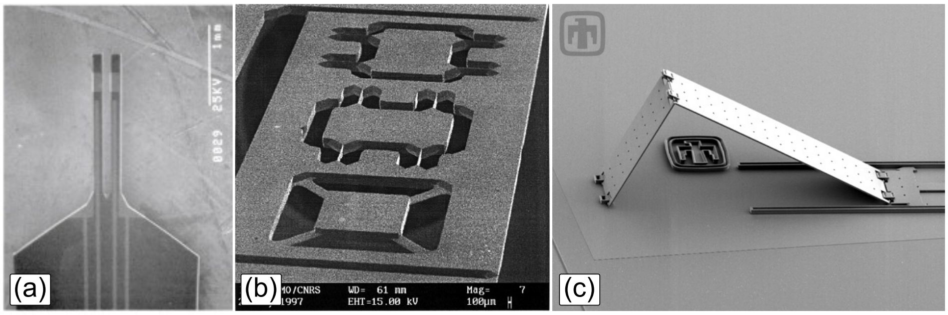


Figure 2.18: Three MEMS structures examples: (a) tuning-fork probe [141], (b) three-dimensional accelerometer [142] and (c) micro-mirror [143].

filters [124, 125], tunable antennas for mobile devices [126–128], RF switches and steerable antennas for radar and aerospace communications [129–131], medical applications like microfluidic cytometer (to measure various parameters of cells) [132], microneedles for drug delivery [133] or specific DNA sequences detection [134, 135], automotive industry applications like airbags detonation, antilock braking system among others [136, 137], and thermo-electric generators used for energy harvesting [138–140].

Fig. 2.18 shows three MEMS example, namely (a) a quartz-based tuning-fork probe with a sharp tip for Atomic Force Microscopy (AFM) systems [141], (b) a three-dimensional accelerometer [142], and (c) a micromirror [143]. Both (b) and (c) are silicon-based MEMS.

Traditionally, MEMS were fabricated on silicon substrates using integrated circuit batch-processing technologies such as surface and bulk silicon micromachining, lithography, etc., due to silicon availability and well-known material properties [144]. This enables the possibility of integrating both silicon electronics and MEMS structures on the same substrate.

MEMS technology still relies on semiconductor industry, however, due to the interest of many major electronic devices vendors and the huge commercialization of these devices[4], materials and technology are constantly expanding and more differentiated of the semiconductor processes [146]. According to this expansion, nowadays alternative substrates such as quartz [141], ceramics [147], plastics [148] and diamond [149, 150] can also be used for fabricating MEMS, although still most MEMS devices are based on silicon as substrate.

[4]The three major MEMS vendors are Bosch, STMicroelectronics and Texas Instruments and they invoice 1011, 970 and 763 US$M respectively due to MEMS devices sales [145].

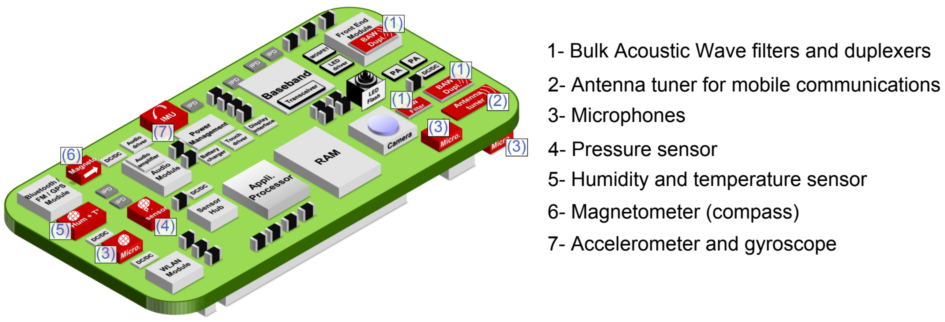

1- Bulk Acoustic Wave filters and duplexers
2- Antenna tuner for mobile communications
3- Microphones
4- Pressure sensor
5- Humidity and temperature sensor
6- Magnetometer (compass)
7- Accelerometer and gyroscope
Baseband
RAM
Appli. Processor
Camera
Power Management
Sensor Hub
WLAN Module
Bluetooth / FM / GPS Module
Audio Module
Front End Module
IMU

Figure 2.19: Schematic of a generic current smartphone. In red, the MEMS devices that can include. Reproduced from [146]

The interest on miniaturized devices has been present since many years ago. Especially important was the physicist Richard Feynman, who was able to foreseeing the miniaturization trend and need arising in semiconductor devices at late fifties [151]. This interest has evolved drastically enabling the possibility of integrating several MEMS devices in many consumer applications such as *smartphones*. A current smartphone contains several MEMS, including motion sensors (gyroscope and accelerometers), wave filters and antenna duplexers, microphones using MEMS technology, etc. Fig. 2.19 depicts the MEMS devices included in a current generic smartphone [146]. Other not so common MEMS included in few devices are an optical image stabilization (that detects phone movement and provides feedback to the autofocus mechanism to compensate for the motion) [152] and an array of micromirrors used for steering light for image projection in an integrated picoprojector [153, 154]. Mainly because of the huge commercialization of smartphones, MEMS sales have been drastically increased in last years and are expected to continue increasing, thus proving the relevancy of MEMS in electronic industry sector [155, 156].

2.4.2 Micromachining processes

There are two main groups of MEMS manufacturing technologies: bulk and surface micromachining. The first one defines structures inside a silicon substrate by selectively etching some parts of it. On the other hand, surface micromachining creates structures on top of a surface by a succession of thin film depositions and selective etching processes.

MEMS can be fabricated by a combination of several processes, including: material deposition, masking specific regions of material and removing parts of material.

The main processes used for depositing thin film material layers can be grouped in those that use chemical reaction and the ones that are based on a physical reaction:

- Due to chemical reaction:
  - *Chemical Vapor Deposition (CVD)*. The substrate is placed inside a reactor where several gases are supplied. Therefore, the substrate is exposed to one or more volatile precursors producing the desired deposit layer on the substrate surface. Many materials can be deposited using this method, including: silicon, carbon nanotubes, silicon dioxide, silicon carbide, silicon nitride and polycrystalline silicon (also called polysilicon) among others. Polysilicon is very used in MEMS surface micromachining since it is easy to deposit and is similar to silicon. On the other hand, silicon dioxide and silicon nitride are used as masking materials in etching processes.
  - *Electro-Deposition (ED)*, also known as *electroplating*. This process is typically restricted to electrically conductive materials. The substrate is placed in a liquid solution. Additionally, a counter electrode (usually platinum) is placed in the solution. Then, a potential is applied between a conducting area on the substrate and the counter electrode resulting in the formation of a layer of the desired material on the substrate. ED is typically used to make films of metals such as copper, gold and nickel in a thickness from approximately $1\mu m$ to more than $100\mu m$.
  - *Epitaxy*. This technology is similar to CVD but if the substrate is an ordered semiconductor crystal (e.g. silicon), the deposited material on the substrate surface keeps the same crystallographic orientation. Similarly, if the substrate is amorphous or polycrystalline the deposited film also has the same atomic structure. One of the most important epitaxy methods is the Vapor Phase Epitaxy (VPE) which introduces several gases in an induction heated reactor where only the substrate is heated enabling the deposition of the desired material. Advantages of this process are the high deposited rate of material and the possibility of producing layers of more than $100\mu m$ thickness.
  - *Thermal oxidation*. This is a simply oxidation process of the substrate surface in an oxygen (O) rich atmosphere. In the oxidation process, atoms of oxygen are combined with the structure of the substrate, which means that the film grows actually downwards into the substrate. This process is limited to materials that can be oxidized and only can form films of materials that oxides the substrate. Furthermore, when the oxide layer is thick enough, the oxygen cannot access the substrate surface and the process is stopped. Thermal oxidation is usually used

to form silicon dioxide layers on a silicon substrate due to it can be used as masking material in subsequent etching processes.

- Due to physical reaction:
  - *Evaporation.* It is a Physical Vapor Deposition (PVD) method to deposit thin films and basically consists in placing inside a vacuum chamber the substrate and the source material to be deposited. Then, the source material is heated to the boil and evaporation point so it is condensed on all surfaces. Many materials can be deposited using this method, however the deposited layer strongly depends on direction and it can result on non-uniform thickness.
  - *Deposition by sputtering.* This technology is also a PVD method, similar to evaporation but particles of source material are ejected using a different method. The process of removing atoms of a material (target) by the impact of energetic particles is known as *sputtering*. The substrate and the target are placed in a vacuum chamber, which is filled with an inert gas (typically argon) at low pressure. Then, a RF power source ionizes the gas and the ions are accelerated towards the target, ejecting surface atoms of the target material which is condensed on all surfaces, including the substrate.
  - *Spin casting.* The material to be deposited on the substrate must be in liquid form or must be dissolved in a solvent. Then, the liquid is deposited by a syringe on the substrate surface which is, then, spun at high revolutions such that the material forms a thin film over the substrate surface once the solvent is evaporated. This is particularly useful in photolithography to apply photoresist to substrates. With this method, single monolayers of molecules can be obtained as well as layers of tens of micrometers.

The commented chemical and physical deposition methods are depicted in Fig. 2.20.

Another important method for MEMS micromachining is the *photolithography*. This process is the transfer of a pattern to a thin film or to a substrate. A typical photolithography process must follow three steps: photosensitive material application, light exposure, and developing. First, a thin film of photosensitive material is deposited on the substrate. Usually, in the MEMS context a photoresist is used as photosensitive material, which is in liquid form and is applied by spin casting. The next step is to expose this material to a pattern of light of a specific wavelength, usually Ultraviolet (UV). The exposure to light causes a chemical change in the photoresist, allowing some zones to be removed by a solution, called *developer*. The third step is to place the deposited photoresist in a developer

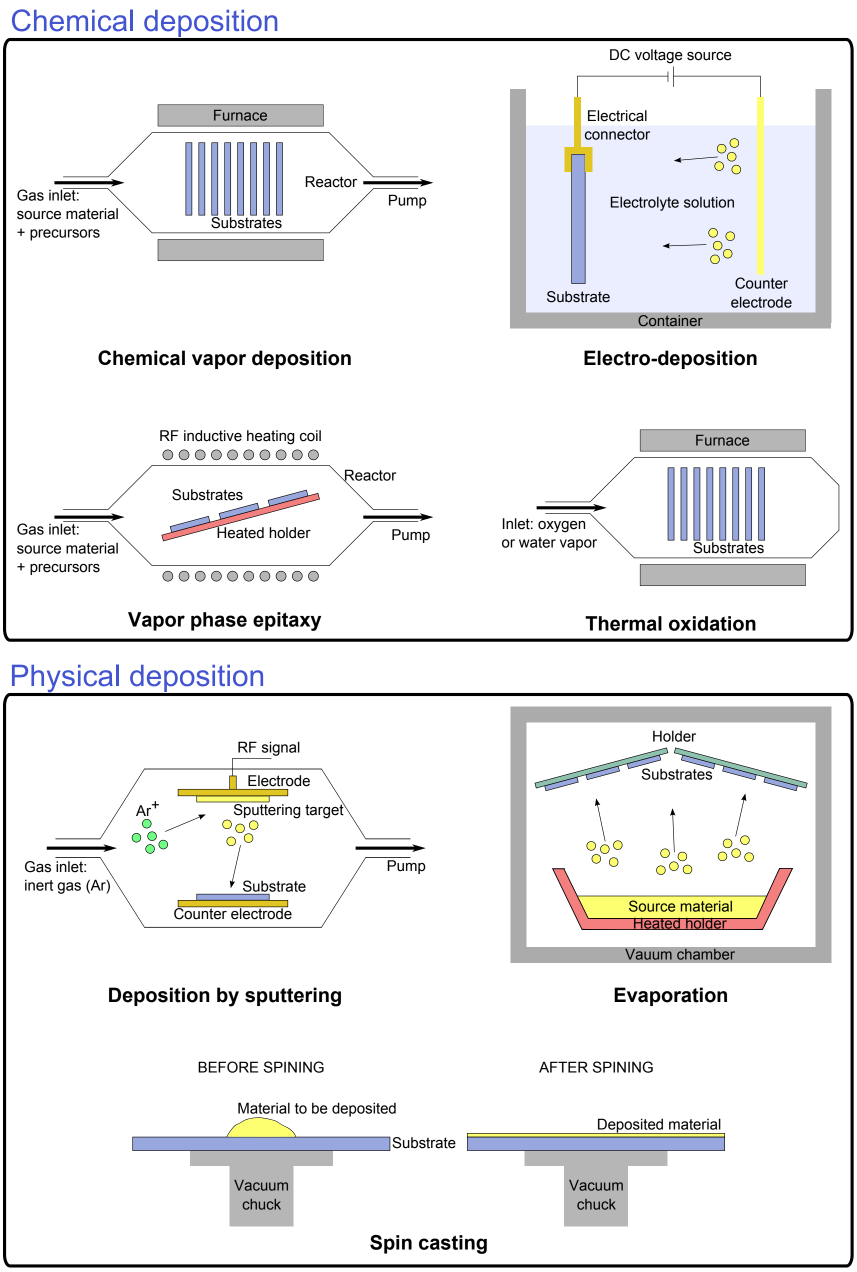


Figure 2.20: Micromachining deposition methods representation.

solution, then, if the resist material is a positive resist, the exposed material is removed and the unexposed zones of the photoresist remain. On the other hand, if the exposed material is resilient and the unexposed material is etched away, the photoresist is considered a negative resist. The typical result after a photolithography process is a thin film pattern of photoresist deposited on a substrate. Then this material can be used as a mask for etching processes, protecting those covered zones of the substrate.

Etching processes can be used to selectively remove those material parts that are not protected with a mask or to completely remove a sacrificial layer, previously deposited, that has been used as support for another layer. Depending on the application and the desired final structure the most proper processes can be used:

- *Wet etching.* Basically it consists in introducing a substrate into a liquid, called etchant, which removes the exposed material. Wet etching processes can be classified in two groups:
  - *Anisotropic etching.* This etching process produces different etch rates that strongly depends on the atomic structure of the material and on the used etchant. This process has been exhaustively studied and can be used for producing complex three-dimensional structures. The modelling of anisotropic chemical wet etching is one of the main topics of the present thesis, thus, it will be detailed in section 2.4.3.
  - *Isotropic etching.* This process is similar to the previous one but different etchants are used. In this case, there is no dependency on the substrate atomic structure and it is etched equally in all directions. Fig. 2.21 depicts a simple comparison of an anisotropic and an isotropic etching processes.
- *Dry etching.* The most common dry etching processes are those that cover a substrate with a plasma. It involves a high-speed stream of glow discharge (plasma) of an appropriate gas mixture being shot in pulses at a substrate. The plasma source acts as an etchant and can be either charged (ions) or neutral (atoms and radicals). The main dry etching processes are:
  - *Ion milling* (or Ion Etching). It is the process of removing atoms by physical sputtering with an inert gas (typically argon). Atoms of the substrate are ejected from the surface by transferring momentum from ions. The systems used are very similar in principle to sputtering deposition systems but now the substrate instead of the target is bombarded by ions. Since usually ions approach the substrate approximately from one direction, this process is highly anisotropic. Moreover, the substrate can be held rotated to modify the direction of impact and masks can be used for etching it selectively.

– *Reactive Ion Etching (RIE)*. This etching process is similar to the Ion milling but when the ions are accelerated towards the substrate, they react at the surface forming another gaseous material that will etch the substrate chemically. Additionally, the sputtering effect of the ions on the surface also contributes to the process of removing atoms from the substrate. Thus, in RIE process there are two contributions, the chemical etching and the physical sputtering. The first one is usually isotropic whereas the physical sputtering is highly anisotropic, hence, by changing the balance between both contributions it is possible to influence the overall anisotropy of the process. Typical results of RIE process are shown in Fig. 2.22. Additionally, sputtering yield (i.e. surface atoms removed per incident ion) may be enhanced significantly compared to pure physical sputtering and Inductively Coupled Plasma (ICP) can be used for increasing etch rate. In the present thesis, an improvement of RIE simulators has been developed thus, this method is detailed in section 2.4.4.

– *Deep Reactive Ion Etching (DRIE)*. A modification of the RIE process in which etch depths of hundreds of microns can be achieved with vertical sidewalls. The original and most used process is based in the so-called *Bosch process* since the Robert Bosch company filed the original patent [157]. This process consists in alternating two different gas composition in a reactor. The first one creates a polymer on the surface of the substrate (passivation), then, the second stage of this method is a RIE such that the polymer layer of horizontal surfaces is quickly removed by the impact of ions. However, the polymer layers of vertical surfaces remain since the polymer dissolves very slowly only in chemical etching. As a result, high aspect ratios can be achieved. For instance, sub-micron deep trenches with an aspect ratio of 160 can be fabricate in silicon with DRIE method [158]. Furthermore, the Bosch process is used for producing through silicon vias of hundreds of microns deep [159, 160]. Due to the alternation of etching and passivation stages, the formation of scallops on the sidewalls is common in the Bosch process [161]. This is depicted in Fig. 2.22. Another DRIE process is the *cryogenic* process which is similar to RIE but the substrate is chilled up to -110 °C such that the chemical reaction (which produces the isotropic etching) slows down while the physical sputtering is still ejecting substrate atoms [162].

These processes are ones of the most used for MEMS micromachining, however, there are many variations and new methods being currently under research [163–165]. By combining different of the listed processes, many current complex MEMS devices are fabricated. An example of a MEMS structure fabrication process is

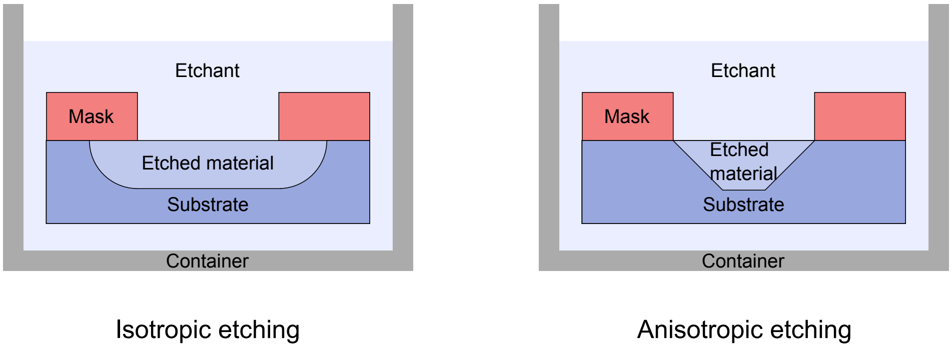


Figure 2.21: Comparison of isotropic and anisotropic wet etching processes.

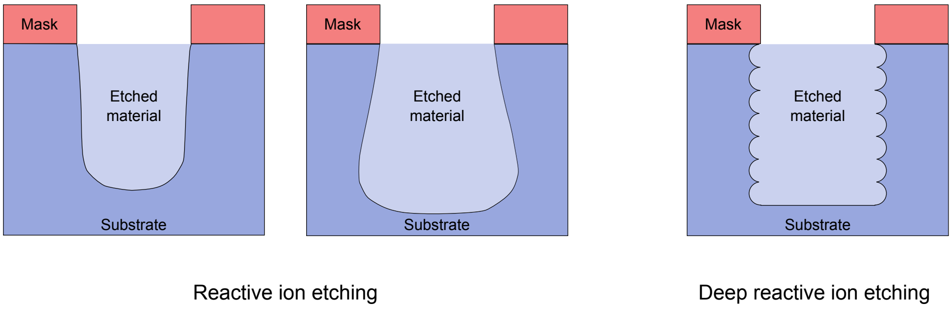


Figure 2.22: Comparison of RIE and DRIE dry etching processes. Two possible results are shown for the RIE example. In the first one the physical sputtering has more influence which leads to almost vertical sidewalls while in the second example, the chemical contribution has more influence and it results in a more isotropic etching. Notice that, in the DRIE example, the scallops have been exaggerated to an easier visualization.

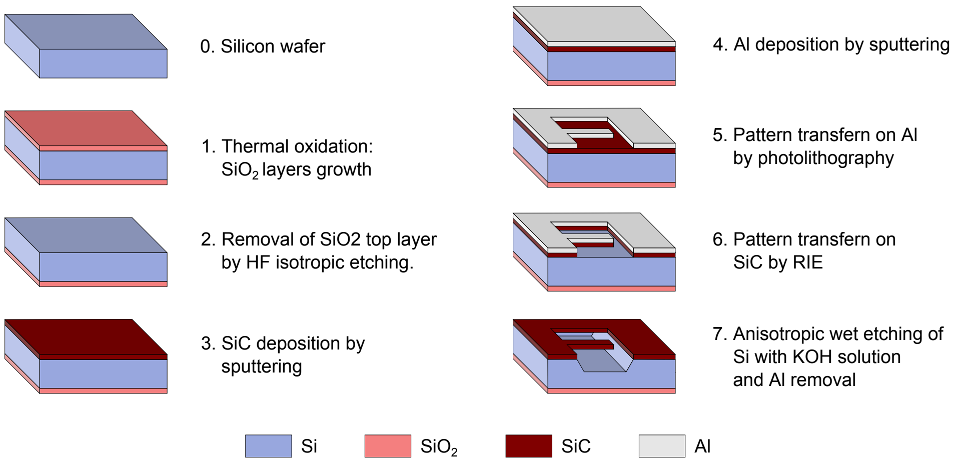

0. Silicon wafer
1. Thermal oxidation: SiO_2 layers growth
2. Removal of SiO2 top layer by HF isotropic etching.
3. SiC deposition by sputtering
4. Al deposition by sputtering
5. Pattern transfern on Al by photolithography
6. Pattern transfern on SiC by RIE
7. Anisotropic wet etching of Si with KOH solution and Al removal
Si
SiO_2
SiC
Al

Figure 2.23: Representation of the steps and methods used for a cantilever micromachining [166].

reproduced in Fig. 2.23. This corresponds to the micromachining process of a cantilever, a very common structure used as RF switch or resonator [166].

2.4.3 Anisotropic wet etching

In this thesis, both the wet etching of silicon and quartz substrates have been simulated by using a LS-based method. One of the main advantages of the LS method is that there is no need to know the crystallographic structure neither the chemicals reactions involved in the etching process due to the macroscopic nature of this method. Nevertheless, a brief explanation of the chemical reactions involved in wet etching process as well as crystallographic structure of silicon and quartz can be helpful to understand the complexity of the process.

The goal of an etching process is to transfer a pattern to a substrate. The first etching processes were used in XV century to etch a metal surface with ornaments. These processes consisted in covering the metal with wax and, then, the desired pattern was created on the wax by removing some parts of it. Finally the metal was immersed in an acid solution such that unprotected areas of the metal surface were chemically removed and the wax pattern was transferred to the metal surface. In this example, an acid solution was used as etchant and wax was used as masking material.

A wet etching process can be isotropic or anisotropic as shows Fig. 2.21. In the first scenario, substrate material is removed at the same velocity in all directions, but, for anisotropic etching, this velocity depends on atomic structure of the substrate. Therefore, the features of an etching process depends on etchant and on material substrate. For instance, due to silicon crystallographic atomic structure, different etchants present different etch rates for each crystallographic orientation. Additionally, there are etchants that remove silicon isotropically, such as a mixture of HF and HNO_3 [167].

In the area of MEMS, wet etching is usually used for creating patterns in silicon and other materials like quartz [168–170]. The first use of anisotropic chemical wet etching on silicon substrates was in the sixties [171] and after that, it was applied to fabricate power transistors [172]. Nowadays, its usage is much extended due to its low cost, the capability of batch processing, and the possibility of fabricating complex structures exclusively with chemical wet etching, such as microprobes [141, 173], accelerometers [142], microneedles [174] or microchannels for microfluids applications [175].

Instead of using acids for silicon etching, alkaline solutions such as potassium hydroxide (KOH), tetramethylammonium hydroxide (TMAH), or Ethylene Diamine Pyrocatechol (EDP) are commonly used [171, 176, 177]. This is a complex process that takes place through sequential oxidation and etching reactions [178]. The oxidation process can be produced by a chemical or by an electrochemical reaction. In the chemical oxidation reaction, surface atoms of silicon that are terminated with an atom of hydrogen (H) are combined with an atom of oxygen terminated with an atom of hydrogen, i.e. H is replaced by OH. Although the hydroxyl group OH^- acts as a catalyst in this stage, the active species is typically H_2O in alkaline solutions. This chemical oxidation reaction can be written as:

$$\equiv \mathrm{Si} - \mathrm{H} + \mathrm{H_2O} + \mathrm{OH^-} \rightarrow \equiv \mathrm{Si} - \mathrm{OH} + \mathrm{H_2} + \mathrm{OH^-} \tag{2.64}$$

Similar reactions occurs for dihydride and trihydride terminated silicon atoms, i.e. with two and one backbond(s) respectively. Furthermore, these reactions are independent of the source of hydroxide solution, such as KOH or TMAH. As can be observed, both a water molecule and a hydroxyl ion are necessary to be close to the replaced H in order to oxidise it. Thus, there are two main factors to determine the reactivity, namely the H terminations and the amount of empty space in the proximity of them. The number of H terminations varies according to the crystal orientation of a silicon, therefore, the oxidation rate, which determine de overall etch rate, depends on the crystallographic orientation of silicon resulting in a highly anisotropic etching [178].

On the other hand, electrochemical studies have found the existence of measurable currents during wet etching, thus suggesting an electrochemical oxidation reaction [179]. In this reaction, also a water molecule is necessary to replace H by OH terminations, however, this does not depend on the amount of OH^- ions. The water molecule directly reacts with a dangling electron that results from a random thermal dissociation of Si-H. This dissociation produces two electrons in the conduction band and leaves enough space for the reaction with a water molecule and it can be produced at any site, which leads to an isotropic etching. The electrochemical oxidation can be written as:

$$\equiv \mathrm{Si} - \mathrm{H} + \mathrm{H_2O} \rightarrow \mathrm{Si} - \mathrm{OH} + 2\mathrm{H^+} + 2\mathrm{e^-} \tag{2.65}$$

Consequently, the overall anisotropy of a wet etching process is determined by the relative importance of the two oxidation reactions.

Both oxidation reactions produce a silicon atom OH-terminated, then, in the etching reaction, this atom is removed from the surface as a $Si(OH)_4$ product. Consequently, the underlying silicon atoms are now H-terminated and ready to be oxidized again. The etching reaction is written as:

$$(\equiv \mathrm{Si})_3\,\mathrm{Si} - \mathrm{OH} + 3\mathrm{H_2O} \rightarrow 3\,(\equiv \mathrm{Si} - \mathrm{H}) + \mathrm{Si}\,(\mathrm{OH})_4 \tag{2.66}$$

Although alkaline solutions have an anisotropic behaviour due to chemical oxidation predominance, the anisotropy depends on several other factors, namely: temperature, concentration of the solution, crystallographic orientation of the substrate, the used etchant and the possibility of adding substances such as alcohols or surfactants.

Quartz substrates has a more complex crystallographic structure than silicon and it is formed by silicon and oxygen atoms such that, an atom of oxygen is always located between two silicon atoms [180, 181]. In a similar manner than silicon, when a quartz substrate is etched with a hydrofluoride solution (often combined with ammonium fluoride NH_4F, such as ammonium bifluoride NH_4HF_2), also presents an anisotropic behaviour [181]. A deeper explanation of the reactions involved in this etching process can be found in scientific literature [182].

2.4.3.1 Crystallographic orientations

As exposed in the previous section, chemical wet etching presents an anisotropic behaviour due to the crystallographic structure of substrates, such as silicon or crystal quartz. This process depends on the used etchant, therefore, the behaviour of an etchant is characterized by the etch rate at each crystallographic orientation.

Silicon atoms are distributed following a Face Centered Cubic (FCC) lattice. This lattice is formed by placing cubic cells contiguously, such that the complete silicon crystal structure is formed by placing the silicon basis (the eight atoms shown in Fig. 2.24 (a)) at every corner of the cubic cells [183]. Fig. 2.24 (b) shows a whole cubic cell, including the atoms that are shared with contiguous cells, which follow the diamond structure. When a whole crystal is cut, different planes and orientations can be obtained according to the corresponding crystallographic structure produced.

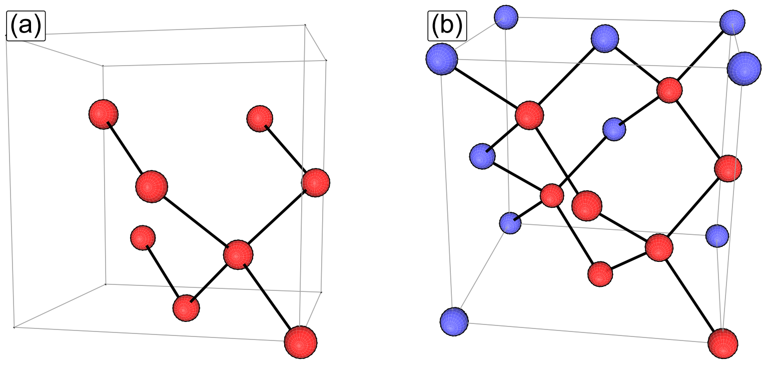


Figure 2.24: Atomic representation of: (a) basis of silicon and (b) diamond FCC structure which also corresponds to silicon. Those atoms in red form part of the basis of the cell, whereas the blue atoms are shared with other adjacent lattice cells.

The most common method to describe atomic planes and directions of a crystal substrate is the Miller indexes. When the atomic structure follows a cubic lattice like silicon, an orientation is defined by a three number tuple. Consider an orthogonal coordinate system, with dimensions x, y, z and unit vector size a. Then, a plane that intersects x-axis at distance a from the origin and it is parallel to y- and z-axis (which is the same that intersecting them at ∞) is defined by $(a/a, a/\infty, a/\infty)$, i.e. (100). That is, Miller indexes of a plane are constructed by taking the reciprocals of the intersect points with the axes. If a plane intersects an axis at some negative point, the negative sign of the point is placed on the number. For example, if a plane intersects axis y at $-a$ and axis x and z at $a/2$, the plane is represented by $(\frac{a}{a/2}, \frac{a}{-a}, \frac{a}{a/2})$, i.e. $(2\bar{1}2)$. Correspondingly, a system of generic planes is defined (hkl), which represents the corresponding perpendicular vectors $\langle hkl \rangle$. In Fig. 2.25 are represented some planes in a cubic cell and the corresponding Miller indexes. Depending on the cut plane (such as the surface of a wafer), a minimal Unit Cell (UC) is defined, which is repeated along the lattice.

Meanwhile, crystal quartz presents a hexagonal system, in contrast to silicon which has a tetragonal system [181, 185]. When Miller indices are used in hexagonal systems, equivalents planes have indices that appear dissimilar. To overcome this, the Miller-Bravais indexing system is used in quartz crystals such that a plane is defined by four indices $(hkil)$, where $i = -(h + k)$. Fig. 2.26 shows the hexagonal system used for analysing quartz crystals (a), and the atomic lattice cell of the α-quartz, formed by silicon and oxygen atoms such that, every oxygen atom is always located between two silicon atoms (b) [180, 186].

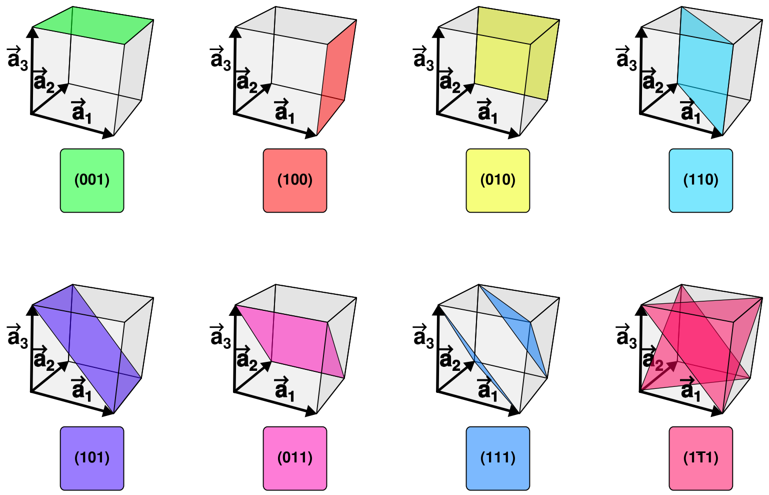


Figure 2.25: Several examples of planes intersecting a cubic cell and the corresponding Miller indexes. Adapted by author from [184].

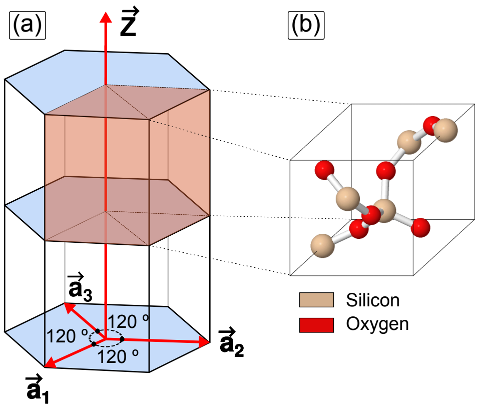


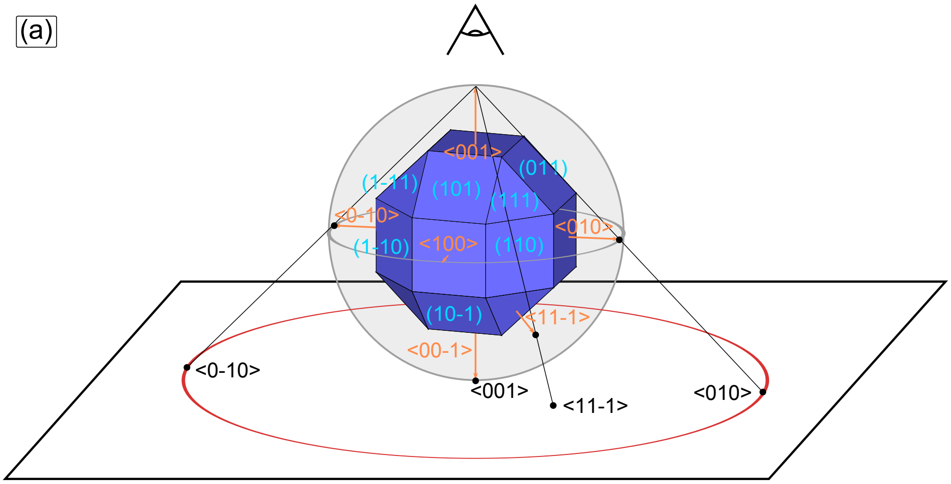
Figure 2.26: (a) Hexagonal system and the four vectors used for defining Miller-Bravais indices. The lattice cell is highlighted with orange. (b) Crystal atomic structure of the quartz lattice cell [186].

2.4.3.2 Etch rate crystallographic orientation dependent

According to etching process, each orientation is etched at a different etch rate due to atomic configuration of crystals. A traditional and useful method to represent the anisotropy of specific etchants and substrate materials is the stereographic projection of a sphere [144]. In stereographic projection, crystal lattice is placed in the center of a sphere and crystallographic directions are projected first onto the sphere and then they are projected onto a plane under the sphere (see Fig. 2.27(a)). This method does not preserve areas but angles are preserved, thus making possible to measure angles between crystallographic orientations. Fig. 2.27 (b) shows two examples of stereographic projections of silicon. Due to crystallographic structure of silicon, any permutation or change of sign of any element of a generic orientation (hkl) results in the same crystallographic structure. Thus, all the possible orientations are contained in the area delimited by the orientations $\langle 100 \rangle, \langle 110 \rangle$ and $\langle 111 \rangle$, which is highlighted in red in Fig. 2.27 (b) [187].

In order to easily visualize the anisotropy of an etchant, the etch rates values of the different orientations are assigned to different colors and stereographic representations are coloured accordingly. A thorough characterization of KOH and TMAH etchants was performed by Sato et al. by etching silicon hemispheres [188, 189]. Nevertheless, another method that consists in etching a silicon structure similar to a wagon wheel was used to characterize KOH [190] and it has recently been proved to obtain accurate characterizations for KOH and TMAH [191]. Furthermore, the usage of additional elements that modify the original anisotropy of the etchant have been studied, including the addition of Isopropyl alcohol (IPA) to KOH solution [192–194] and the addition of surfactant Triton X-100 to TMAH solutions [195]. A complete characterization of KOH, TMAH with and without additives was performed by Gosálvez et al., including results at several temperatures and etchants concentrations [191]. A simulation result of a wagon wheel experiment and the reconstructed etch rate stereographic projection of KOH 30 wt% at 61.2 °C are shown in Fig. 2.28 [191].

Although the crystallographic structure of quartz is more complex than silicon structure, the same stereographic projection technique is used for visualizing the different etch rates depending on crystallographic orientations. The main planes presented by a quartz crystal are shown in Fig. 2.29 (a). Although first experiments to characterize the wet etching process of quartz were performed mainly with Z-cut quartz specimens [169, 181], the etching of quartz hemispheres has been successfully used for obtaining the etch rate distribution of various etchants according to crystal orientations [180, 185, 196]. An initial hemisphere quartz and the resulting hemisphere after an etching process with NH_4HF_2 solution are shown in Fig. 2.29 (b) and (c) respectively. In addition, the reconstructed etch rate stereographic projection of this etchant is shown in 2.29 (d). An accurate



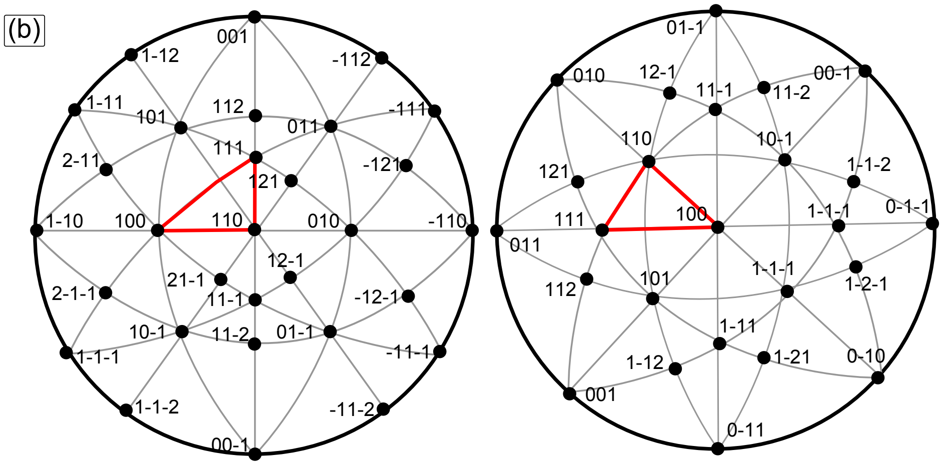


Figure 2.27: (a) Stereographic projection technique of a (001)-oriented silicon crystal. (b) Stereographic projections of a (110)- and (100)-oriented silicon crystals.

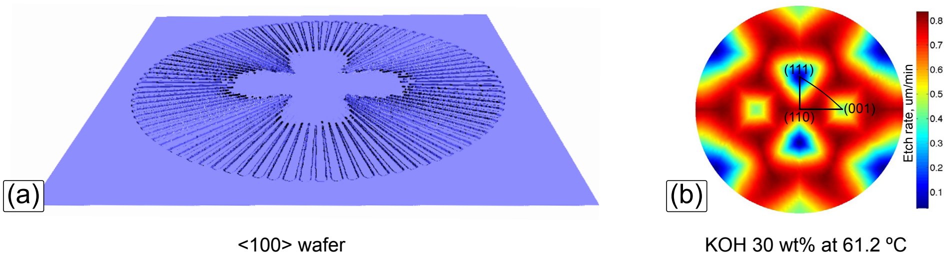


Figure 2.28: (a) Simulation result of a wagon wheel experiment used for obtaining etch rate distribution of KOH 30 wt% solution at 61.2 °C (reproduced from [187]). (b) Stereographic projection of the etch rates (reproduced from [191]).

characterization of NH_4HF_2 as anisotropic etchant of quartz was performed by Cheng et al. by etching a hemisphere [196].

2.4.3.3 Simulation of anisotropic wet etching

Anisotropic chemical wet etching is a complex process that depends on many factors, such as etchant solution (e.g. KOH or TMAH), substrate material (silicon or quartz among others), temperature of the solution, and the pattern mask applied to the substrate surface protecting specific regions against etching. Consequently, it is not trivial to predict the result of a concrete scenario and many experiments had to be performed in order to finally obtain the desired structure. Of course this methodology would imply a waste of time and resources, thus, the development of accurate simulators has been of interest for the scientific community since many years ago.

The first wet etching simulators strategy is known as *geometric simulators* since the anisotropic etching is emulated by planes which are evolved at the corresponding velocity in the normal direction [198]. Therefore, the velocity of every orientation was needed in order to simulate the process accurately [199]. Several geometric models were presented and were able to emulate properly the anisotropic etching [200–202], nevertheless, there are to factors that encourage the research of different simulators: the complex data base needed of every orientation and the high computational effort required by these models, specially to compute the interaction of several planes like etching of both top and bottom sides of a wafer [203].

As an alternative, simulators based on an atomistic strategy were introduced. This approach describes the crystal substrate as a collection of cells or atoms. Each atom is connected with the neighbouring atoms according to the crystallographic orientation, as can be visualized in Fig. 2.30 for some silicon orientations. The etching process in these models is emulated by removing selectively the surface

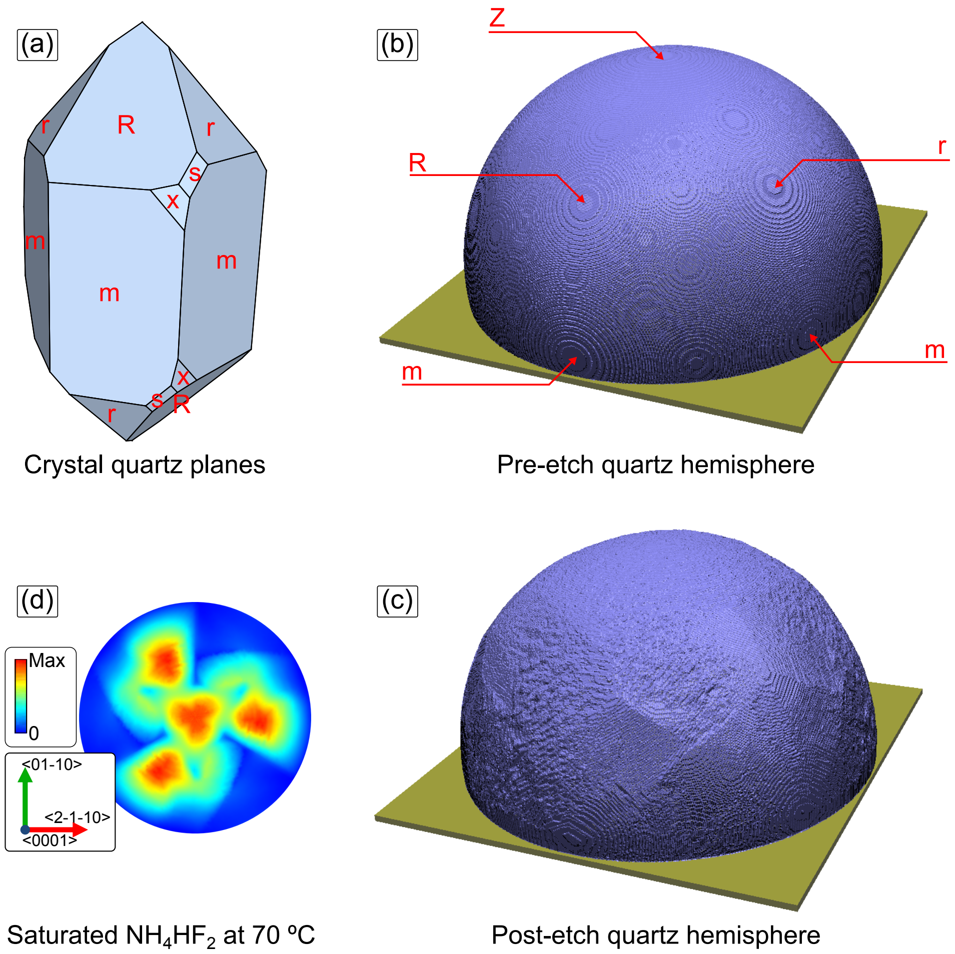

(a)
R
r
r
s
x
m
m
m
m
x
s
R
r
Crystal quartz planes
(b)
Z
R
r
m
m
Pre-etch quartz hemisphere
(d)
Max
0
<01-10>
<2-1-10>
<0001>
Saturated NH_4HF_2 at 70 ºC
(c)
Post-etch quartz hemisphere

Figure 2.29: (a) Planes of a quartz crystal. Traditional notation of the different planes according to geometrical shape is shown. In (b) and (c), a pre- and post- etch quartz hemispheres are shown (obtained with Intellietch [197]). The etching process is performed using NH_4HF_2 as etchant. The final etch rate distribution is presented in (d) (obtained with Intellietch [197]).

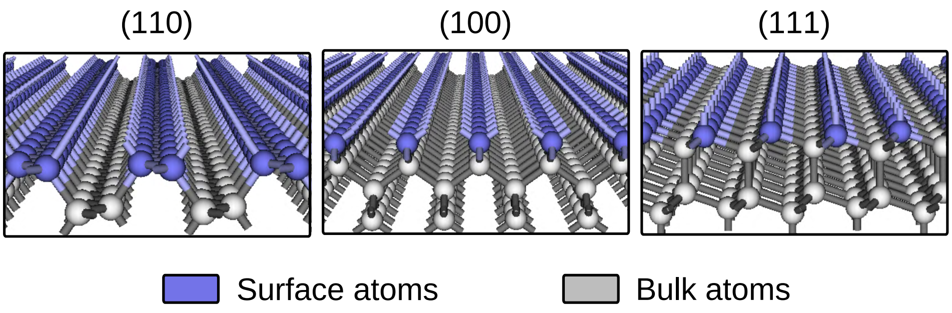


Figure 2.30: Connections between surface and bulk atoms (blue and white respectively) in atomistic models for silicon orientations (a)⟨100⟩, ⟨110⟩ and ⟨111⟩. Obtained with visualTAPAS [204] and reproduced from [187].

atoms. The atoms of the surface are inspected to determine the removal rate according to their neighbourhood configuration, deciding whether the atom is removed or remains at the surface.

Within the atomistic strategy, two schemes can be distinguished, namely the Kinetic Monte Carlo (KMC) and Cellular Automata (CA). In the KMC, each atom has a different probability depending on its orientation and then, an atom is randomly selected to be removed from the surface such that those atoms belonging to an orientation with a higher etch rate have a higher probability to be removed and vice versa. Hence, when an atom is decided to be removed it is immediately removed. This method allows a suitable tool for the exploration of the surface morphology [205, 206] and the simulation of complex MEMS structures [207]. The removal probability can be obtained by theoretical considerations of the chemical reactions [208] and by comparing with experimental results, which usually implies a complex calibration process [209–211].

Another atomistic approach is the CA. A CA system is a lattice of connected atoms, such that each of them is labelled with a state. Depending on the current state and the neighbouring atoms, the states are updated after each time step by applying the same set of rules. The first simulators based on CAs are the discrete CAs applied to wet etching of silicon [212]. This method removes every atom surface but the surface atoms with three bulk neighbours, i.e. (100) surfaces. This simple method can model properly some etchants like KOH but it is very limited when trying to emulate other etchants or complex surfaces [203].

Another simulator based on CAs is known as stochastic CA [212]. Similarly to KMC, each atom was assigned with a removal probability depending on the surface orientation. The possible atom states are only *occupied* or *empty*, such that an occupied atom belongs to the bulk and an empty atom has been etched away [213–

215]. The main difference with KMC models is that, while KMC operates purely sequentially, the CA approach operates with a *decide and perform* strategy, i.e. a first iteration is used for deciding which atoms have to be removed depending on the removal probability of each one and, then, a second iteration removes the selected atoms and updates the state and the removal probability of their neighbours. Due to the randomness of this process, usually multifaceted surfaces at convex corners are not well reproduced.

Zhu and Liu introduced a new scheme based on CA, namely the Continuous Cellular Automata (CCA) [216]. The CCA avoids the random noise produced by stochastic CAs. In this method, each atom configuration is associated with a specific etch rate and have an *occupation* value between 0 and 1. Initially, every atom has an occupation 1, then, the etching process is emulated by gradually reducing the occupation of the surface atoms as indicated by their associated etch rates. When an atom reaches an occupation ≤ 0, it is removed from the bulk. Due to the *decide and perform* strategy, identical atoms are removed simultaneously.

These atomistic models reproduce a macroscopic behaviour (surface evolution) by the definition of microscopic rules. Many effort has been put by researchers to link both levels and obtain accurate simulations. Accordingly, a thorough classification of silicon atoms that allowed the correct emulation of complex surfaces was presented [217]. Additionally, the effect of physical phenomena enabled to obtain an accurate set of etch rates for many crystal orientations [218, 219]. Moreover, the CCA has also proved to properly simulate the usage of additives in silicon wet etching [220], design of cavities and mesas with sharp corners [221], and quartz etching with ammonium bifluoride [180].

Despite the good agreement with experimental results, an intrinsic drawback of atomistic models is that the final result is represented as a cloud of points and it makes difficult to visualize some complex geometries. Accordingly, in chapter 3 of this thesis this problem is addressed.

Although the CCA represent a significant improvement respect with geometric simulators regarding accuracy and required experimental etch rates, this atomistic method still requires a laborious calibration process to relate experimental macroscopic etch rates with atomistic configurations of the different atoms in the lattice. Optimization algorithms are the most common for calibrating the CCA. Basically, these algorithms start with an initial set of parameters, run a simulation of a specific experiment, and the error between specific measurements of experimental and simulated data is calculated. Then, the initial parameters are modified and these steps are repeated until the measured error is low enough. The calibration process of the CCA is a complex task that can require tens of hours to converge and must be performed in order to simulate any etchant or any configuration change such as different temperature or concentration [180, 222].

Consequently, several implementations based on the LS method has been presented in last years since this calibration process can be avoided [223–226]. In chapter 4 the LS implementation developed for wet etching of silicon and quartz is explained.

2.4.4 Reactive Ion Etching

In wet etching processes, a liquid solution is used as etchant, conversely, in dry etching processes a gaseous environment, usually in vacuum, is used as etchant [227]. The RIE is a dry and plasma etching process in which both chemical and physical effects contribute to remove material from the substrate. The etch rate of this process is substantially higher than purely physical processes like ion milling, in which only physical sputtering effect takes part in the etching process. One of the most characteristic features of RIE is the capability of highly anisotropic etching without relying on crystallographic orientations of the substrate material, in contrast with wet etching which strongly depends on crystal planes (see section 2.4.3.2).

For RIE process, the substrate is placed between two parallel plates inside a reactor in which both plasma and DC bias voltage are created using a Capacitively Coupled Plasma (CCP). The substrate is situated on one of the electrodes which is electrically isolated, whereas the other one is connected to ground. A strong RF signal (typically 13.56 MHz at a few hundred watts) is applied between the two electrodes in order to generate a plasma from the selected etch gases. Due to the high mobility of electrons, this RF signal makes them impact with the substrate platter electrode. Thus, due to its DC isolation, a large negative voltage is created [228]. On the other hand, ions have low mobility and the plasma acquires a positive voltage as depicted in Fig. 2.31(a). This DC bias voltage between the plasma and the electrode produces ion bombardment towards the electrode, exposing the substrate to heavy ion bombardment. The ions react chemically with the surface of the substrate due to neutral species stuck at the surface but also can remove material by physical sputtering. Due to chemical reactions, the etch yield is substantially increased which is known as *ion enhanced etching* [229].

Additionally to etching produced by ion bombardment, chemical isotropic etching is also produced at the substrate surface. Nevertheless, the overall etching behaviour of RIE process is an anisotropic etching independently of the crystallography of substrate material. This anisotropic behaviour is obtained thanks to two factors: the normal direction of ions that mostly impact only on the horizontal surface of the substrate and the formation of a sidewall passivation layer that reduces or completely stops the lateral etching or *underetching* produced by chemical reactions [230]. This passivation layer is formed by redeposited etched mask atoms and inert species, including species from the etchant gas solution condensed in vertical walls, or non-volatile etch products [231]. This anisotropic process is depicted in Fig. 2.31(b). As it is shown, the mask material is also

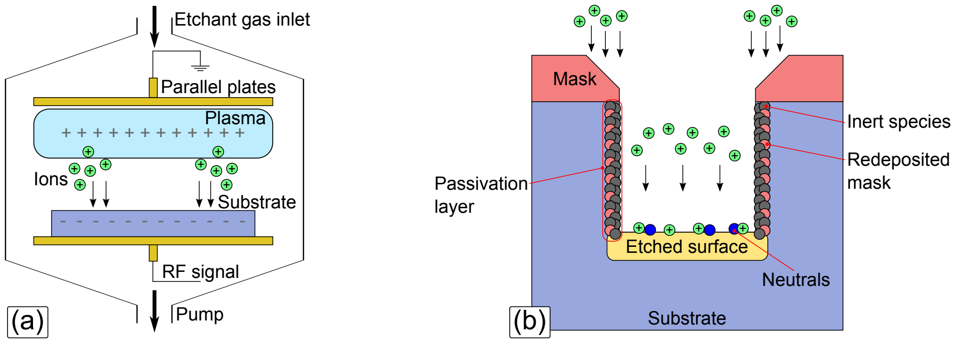


Figure 2.31: RIE schematic setup (a) and anisotropic etching produced by verticality of ions bombardment and the formation of a sidewall passivation layer (b).

etched away due to physical sputtering of ions. Hence, selectivity[5] of silicon to photoresist masks is usually 1-10:1 [232].

Etch rate and anisotropy are consequence of balancing etching and deposition reactions, which is usually done by selecting properly the concentration of etchant solution or using additional gases [231]. Consequently, RIE can produce perfectly vertical walls, fully isotropic etching and positively or negatively tapered walls [232] (as shown in Fig. 2.22).

Many materials can be etched using RIE processes, including silicon, silicon dioxide (SiO_2), polysilicon, and silicon carbide (SiC). Relying on the target material, different gaseous solutions that react with the material must be used, for example, silicon can be etched in fluorine, chlorine or bromine plasmas such as $SiF_4, SiCl_4$ or $SiBr_4$ [233]. On the other hand, silicon dioxide is normally etched with C_2F_6, C_3F_8 or CHF_3 [232].

Many parameters like RF power, bias voltage, chamber pressure, temperature, etch gas flow rates, and the usage of different gas mixtures will determine the etching behaviour, namely etch rate, selectivity, sidewall angle or other responses [234, 235].

In comparison with wet etching, RIE is economically much more expensive due to equipment requirement. Some materials like silicon or silicon dioxide, can be etched by both methods, thus, when deciding which process must be used in a micromachining experiment, the usage of a RIE must be justified. However

[5]The term *selectivity* denotes the capability of removing the target material without removing other materials. For instance, the capability of removing the substrate while preserving the mask material that protect specific regions of the substrate. Thus, a high mask selectivity is suitable for etching processes.

there are materials that only can be etched with RIE, like silicon carbide and titanium carbide. Each process has advantages over the other, for instance, RIE process presents a higher etch rate, photoresist masking works well for many RIE applications, etching only one side of substrate needs no protection on backside, high aspect ratio can be fabricated simply, and sidewall angle can be adjusted by modifying parameters of the experiment. On the other hand, wet etching is a simpler process that allows a high throughput in batch mode, it is easy to work with any wafer shape and size, and the resulting surface is less damaged than with RIE process [232].

Due to the good features that present both processes, usually they are combined in MEMS micromachining to produce complex structures, like the example shown in Fig. 2.23 in which SiC is etched with a RIE process using aluminium as masking material and a KOH solution is used for etching a silicon wafer following the pattern created on the SiC. Many MEMS examples that use both methods can be found, like prismatic beams [236] and torsional resonators [237].

2.4.4.1 Simulation of Reactive Ion Etching

RIE is a complex process that relies on many parameters, such that process output depends on many externally controlled variables. Additionally, due to the complex physical and chemical processes involved in RIE, many non-ideal effects occur in experiments, including undercutting of mask due to small passivation layer failures [238], undercutting of the sidewall at the bottom corner of a trench [239, 240], and high aspect ratio structures can be etched more slowly due to a flux reduction of the arriving ions (RIE lag effect) or even faster because the amount of passivation species that arrives at the bottom is decreased (inverse RIE lag effect) [241, 242]. Originally, trial-and-error strategies were used until the desired result was achieved. Nevertheless, as more accuracy was required by industry, plasma etching simulators capable to predict both, the plasma features and the etching process properties, became fundamental [243]. These tools allow to emulate the effect of varying several external parameters of an experiment [244].

One of the main issues in modelling plasma etching is the wide range of physical measurements and temporal scales, ranging from atomistic measurements like removed atoms, to macroscopic dimension such as reactor and substrate sizes [243]. Accordingly, a typical strategy to deal with this problem in RIE simulation is to divide the process into two different domains, namely the reactor-scale and the feature-scale.

The reactor-scale domain provides the incident particles fluxes, yields, energy and angular distribution relying on equipment and experimental scenario (pressure, bias power, gas flow and concentration, etc.). This data can be provided by equipment simulations or by information obtained directly from experiments [245–

247]. This reactor-scale data is then used as a data base by the feature-scale simulator.

The feature-scale simulation tool is commonly divided in three main stages:

- *Flux calculation*: given a parametrized structure, this stage calculates local fluxes of the different species at each face of the structure being etched. The visibility and orientation of the faces are considered in this stage. There are mainly two methods to calculate the local fluxes, the first one consists in integrating the fluxes (provided by reactor-scale domain) over the solid view angles [248, 249]. The other way is by performing Monte Carlo simulations of the particles impacting on the structure according to the data provided by reactor-scale domain [246]. This simulated particles can be absorbed, thus, changing the surface composition (for example forming a passivation layer that reduces the etch rate) or can be reflected and reemitted [250].

- *Etch rate calculation*: this stage calculates the local etch rate of each face being etched according to the species local fluxes previously calculated, the yields of the particles and the composition of the surface [251, 252].

- *Profile evolution*: According to the local etch rates, this stage updates the surface of the structure being etched. There are several methods for evolving the surface. The *string method* and the improved *method of characteristics* represents the surface as a string of nodes connected by straight lines [253]. Although they are simple algorithms, they present problems at sharp corners [254]. Another evolution technique is the *cell-based method* [255], which is equivalent to the CA representation of section 2.4.3.3. This technique is well combined with Monte Carlo simulations [256]. Although cell-based method provides excellent results, promising LS implementations have been presented in last years. Kokkoris et al. presented a three-dimensional LS-based RIE simulator of silicon dioxide and DRIE of silicon [257]. This model was able to reproduce the lag and inverse lag effect. On the other hand, Radjenović et al. used a simpler fluxes model but introduced the usage of the SFM for plasma etching [258]. Ertl and Selberherr presented a LS implementation for deposition, RIE and DRIE that efficiently used the SFM in combination with RLE to reduce computational effort and memory usage [259, 260]. The advantages of the LS method is the trivial formation of sharp or smooth surfaces, however, usually an explicit parametrization of the surface is required to calculate local fluxes, therefore, the implicit surface must be extracted from implicit function.

All of these modules must be executed iteratively to advance progressively the surface since fluxes and surface composition change with the etched surface topography. Notice that some simulators include the reactor-scale simulation in this iterative process such that, the new profile of the structure is considered for the calculation of the fluxes [250].

In chapter 5 the implementation of a LS module for profile evolution developed in the present thesis is detailed. This module is used in combination with the software Anetch developed by the Fraunhofer IISB [261]. This tool emulates the RIE of silicon dioxide in C_2F_6 gas solution.

Chapter 3

Improvement of the visual representation of atomistic wet etching simulators by means of the Level Set method

This chapter presents an application of the LS method to improve CCA-based simulators results oriented to chemical wet etching process. Since these results are sets of unconnected atoms, the goal of the LS implementation is to obtain a continuous surface from these points in order to improve the results visualization.

First the visualization-related drawbacks of the CCA are introduced in section 3.1 as well as the motivation for the implementation of a LS method to improve CCA results. Then, section 3.2 explains the two different models used, including the required numerical techniques. In this section, the developed algorithms required by the two models are also described. Later, section 3.3 collects the details of the two developed implementations, namely, an original LS implementation that updates the whole three-dimensional space and a local LS implementation based on the SFM. Additionally, this section includes the algorithms of both implementations. In section 3.4 several examples are studied and results of both implementations are compared and characterized against CCA results. Finally, conclusions are presented in section 3.5.

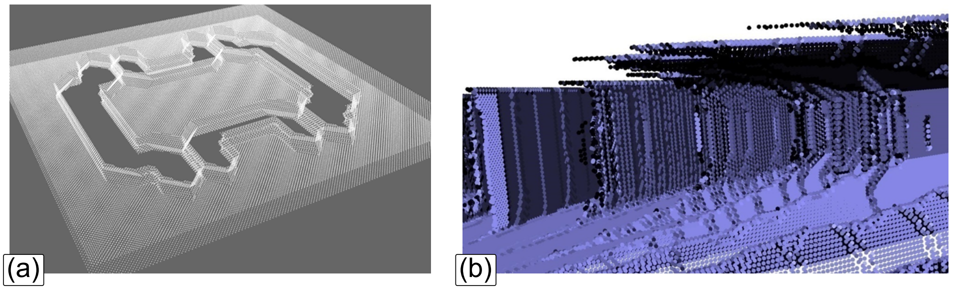


Figure 3.1: Two results of a CCA-based wet etching simulator proving the difficulty of visualization: (a) uncoloured cloud of points and (b) coloured points according to their normal vector. Images obtained with Intellietch [197].

3.1 Introduction and drawbacks of cellular automata simulators

The most currently used simulation methods for MEMS micromachining processes are based on atomistic approaches. In particular, the CCA is commonly used for simulating the anisotropic wet etching process. By continuously removing surface atoms, the CCA emulates the etching process, thus, resulting in a set or cloud of atoms. The resulting three-dimensional structure depends on many factors such as the etchant, temperature, mask pattern used to protect specifics regions of the substrate, and the crystallographic orientation of the material. These dependencies allow to obtain many different complex structures, containing both sharp and smooth surfaces.

Despite of accurate results obtained with the CCA, final results are a cloud of unconnected points because of intrinsic atomistic nature of the CAs. Therefore, it is hard to visualize correctly some details of structures, especially when dealing with complicated topologies as shows Fig. 3.1(a). In addition, the CCA approach introduces some noise due to the calibration process required for obtaining the atomistic etch rates from experimental macroscopic measurements [222]. As a consequence, a robust technique must be applied in order to ease visualization and understanding of the structures.

The method currently used for improving the final visualization (for example in Intellietch commercial software [197]) is to shade the points depending on their normal vector. Unfortunately, in complex morphologies of some structures the shading accuracy of the method is not good enough and adds too much noise, so the visualization quality decreases greatly (see Fig. 3.1(b)). Thus, in order to improve the visualization, it is necessary to obtain a continuous surface from the information of the remaining atoms. This process is known as image or surface (in three dimensions) reconstruction and, when applied to MEMS structures must be sufficiently versatile to reconstruct all the different kinds of topologies that can be obtained with anisotropic wet etching.

Accordingly, in this thesis the usage of a LS algorithm for image reconstruction from scattered points is proposed.

3.2 Image reconstruction with the Level Set method

There are many methods for surface reconstruction [262–266]. In particular, triangulation methods such as Delaunay triangulations and Voronoi diagrams are very popular [267–269]. This triangulation technique generates a mesh from unconnected points by connecting the points with their adjacent points, forming triangles that fulfil the Delaunay condition, i.e. the circumferences defined by the three points of all the triangles have no other point within their interior. Hence, every point is connected with its neighbouring points and a continuous surface can be reconstructed.

Although these techniques can provide suitable results in many applications, they are not adequate for noisy data and data that can present holes or boundaries [270]. Furthermore, they generally require a constant density of points and, CCA results of wet etching simulations usually present complex parts that are formed only by a few atoms and structures with holes, especially when etching simultaneously the top and the bottom of a substrate.

On the other hand, the LS method has proven to be robust even with noisy data that is formed by different densities [271, 272]. In addition, the LS approach can handle the splitting and joining of several surfaces with no additional effort, which result very valuable when is applied to MEMS structures. Another advantage of the LS with respect to triangulation methods is the possibility to generate non-linear surfaces.

The LS is a technique to evolve a front (surface in three dimensions) according to some physical fields or properties. The main idea of the LS when applied to three-dimensional surface reconstruction is to start with an initial surface such that it surrounds all the scattered points. Then, the initial surface is evolved up to the points by the LS method, thus preserving the continuity of the surface. A simple two-dimensional example of this process is depicted in Fig. 3.2.

On the other hand, the high computational cost of the LS method can be drastically reduced by applying local techniques like the SFM and implementing a parallel algorithm to be executed on a GPU.

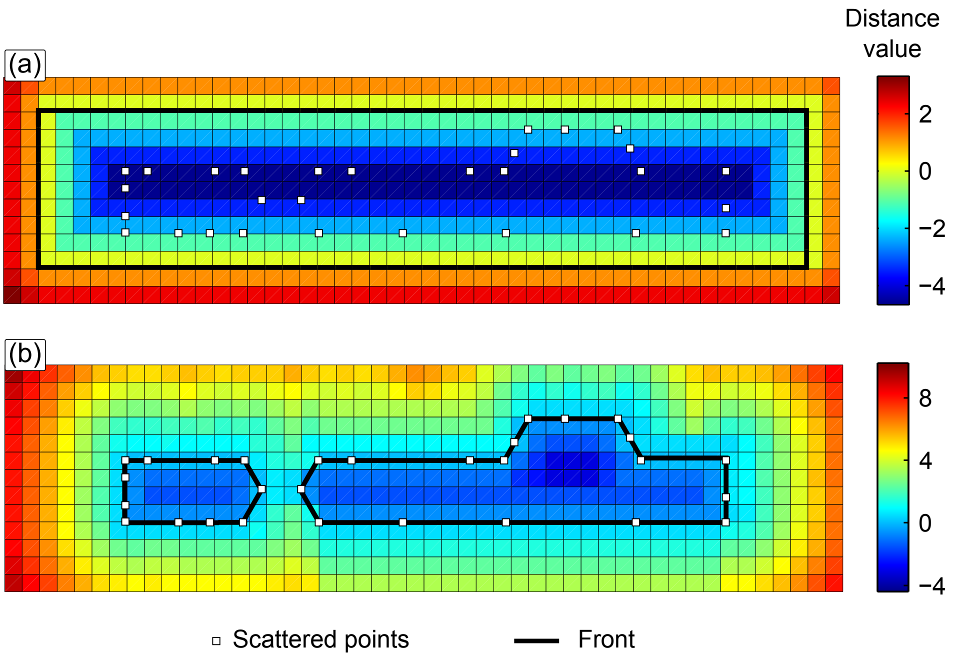


Figure 3.2: Simple two-dimensional example of image reconstruction from scattered points. The front and the corresponding SDF are shown: (a) initial front, (b) evolved front adapted to the scattered points.

3.2.1 Minimal surface energy model

Despite some models for LS image reconstruction have been presented [7, 273], in this thesis the minimal energy model presented by Zhao et al. [272] has been chosen since this model does not need any additional information, only the data points (i.e. atoms of CCA results, from now on referred as set S), and also because of the complex structures that result from wet etching simulations. Although some information from the atomistic configuration, like neighbouring atoms connected to a specific atom, could be used as input parameters to improve the reconstruction process, CCA models usually produce noisy data as well as isolated atoms (see Fig. 3.1(b)). Therefore, a model which only requires scattered points to reconstruct the surface has been chosen.

This energy model defines an energy function of a surface which is proportional to the sum of the distance values of those points that form the surface. This energy function is defined as:

$$E(\Gamma) = \left[\int_{\Gamma} d^m(\vec{x}) ds \right]^{1/m}, \tag{3.1}$$

where Γ is a surface, ds the surface area, $m \in [1, +\infty)$ variates the smoothness of the surface, and

$$d(\vec{x}) = \sqrt{(x - x_S)^2 + (y - y_S)^2 + (z - z_S)^2} \tag{3.2}$$

is the unsigned distance between surface points $\vec{x} = (x, y, z)$ and the S points $\vec{x_S} = (x_S, y_S, z_S)$.

The purpose of this model is to find a surface adapted to S points, which corresponds to a local minimum of surface energy. Therefore, initial surface is evolved following the gradient descent of the energy function:

$$\frac{\partial E(\Gamma)}{\partial \Gamma} = \frac{1}{m} \left[\int_\Gamma d^m(\vec{x}) ds \right]^{\frac{1}{m}-1} \left[m \cdot d^{m-1}(\vec{x}) \nabla d(\vec{x}) \cdot \vec{N} + d^m(\vec{x}) \kappa \right], \tag{3.3}$$

where $\vec{N}$ is the normal vector of the surface and κ the mean curvature. When the minimal energy is achieved, (3.3) turns out:

$$d^{m-1}(\vec{x}) \left[\nabla d(\vec{x}) \cdot \vec{N} + \frac{1}{m} d(\vec{x}) \kappa \right] = 0. \tag{3.4}$$

Because of the balance between both terms, $\nabla d(\vec{x}) \cdot \vec{N}$ and $d(\vec{x})\kappa$, of (3.4), the reconstructed surface is more flexible in high sampling density regions and more rigid in regions with a low sampling density [271]. There are two minima that satisfies (3.4), the first one is the trivial solution $\Gamma = \emptyset$ and the second one $\Gamma = \Gamma_0$ is the desired surface attached to S points. The surface Γ_0 will approximate the real and desired shape of the S cloud if the sampling density of data is enough to resolve the real shape. In two dimensions, the local minimum corresponds with a polygon connecting adjacent points by straight lines, however in three dimensions, the minimal surfaces is smoother and it has no edges [272].

In order to achieve the reconstructed surface that corresponds to the minimal energy of (3.4), an initial surface Γ that encloses all the S points is evolved in the normal direction according to the energy gradient flow of (3.3), leading to:

$$\begin{aligned} \frac{\partial \Gamma}{\partial t} &= -\frac{\partial E(\Gamma)}{\partial \Gamma} \vec{N} \\ &= -\left[\int_\Gamma d^m(\vec{x}) ds \right]^{\frac{1}{m}-1} d^{m-1}(\vec{x}) \left[\nabla d(\vec{x}) \cdot \vec{N} + \frac{1}{m} d(\vec{x}) \kappa \right] \vec{N}. \end{aligned} \tag{3.5}$$

In this study, $m = 1$ is used since it allows to obtain accurate results and is the less computational costly. Accordingly, (3.5) turns out:

$$\frac{\partial \Gamma}{\partial t} = -\left[\nabla d(\vec{x}) \cdot \vec{N} + d(\vec{x}) \kappa \right] \vec{N}. \tag{3.6}$$

When the surface is far away from S, both terms make the surface shrink and finally the equilibrium of (3.4) is achieved, resulting in:

$$\nabla d(\vec{x}) \cdot \vec{N} = -d(\vec{x}) \kappa. \tag{3.7}$$

When the surface is close to S points, $\nabla d(\vec{x})$ and $\vec{N}$ tend to be perpendicular and the left side of (3.7) tends to zero, thus, a small distance value $d(\vec{x})$ is required to fulfil the condition, which corresponds to those points attached to S points.

The election of a proper initial surface is important since, if it is too far from S, the surface may need many steps to reach the solution, which requires a high computational effort. Additionally, in complex topologies, a different local minimum energy could be found and the desired surface would not be reached. In practice, a good initial surface is formed by a contour of the distance function, i.e. $d(\vec{x}) = \epsilon$, being ϵ a constant. In section 3.2.5, an algorithm to find such initial surface is described.

Once the velocity of the surface is defined by (3.6), the surface is embedded in the zero contour of the ϕ function:

$$\Gamma(t) = \{\vec{x} : \phi(\vec{x}, t) = 0\}. \tag{3.8}$$

Then, the front velocity is extended to every level and the chain rule as well as the LS formulation are applied, leading to:

$$\frac{\partial \phi(\Gamma(t), t)}{\partial t} = \frac{\partial \phi}{\partial t} + \frac{\partial \Gamma(t)}{\partial t} \cdot \nabla \phi = 0. \tag{3.9}$$

Hence, by taking (3.6), the LS equation is obtained:

$$\frac{\partial \phi}{\partial t} = \left[\nabla d(\vec{x}) \cdot \vec{N} + d(\vec{x})\kappa\right] \vec{N} \cdot \nabla \phi, \tag{3.10}$$

which can be written as:

$$\phi_t = \left[\nabla d(\vec{x}) \cdot \vec{N} + d(\vec{x})\kappa\right] |\nabla \phi|. \tag{3.11}$$

The application of the LS method for evolving the surface allows to take advantage of all the benefits explained in sections 2.1 and 2.2, like trivial handling of topological changes and formation of sharp and smooth surfaces.

3.2.1.1 Numerical schemes

The LS equation of the energy model (3.11) can be developed to the form

$$\phi_t = \nabla d(\vec{x}) \cdot \nabla \phi + d(\vec{x})\kappa |\nabla \phi|, \tag{3.12}$$

which is a convection-diffusion equation like (2.59) (studied in section 2.2.3), where $\vec{V} = \nabla d(\vec{x})$ corresponds to a completely external velocity field and the second term corresponds to a local curvature dependent motion with $b = d(\vec{x})$. Each term must be solved with the corresponding numerical techniques in order to ensure convergence and numerical stability. Upwind differencing technique can be used to solve the first term. Thus, taking into account that

$$\nabla d(\vec{x}) = \left(\frac{\partial d(\vec{x})}{\partial x}, \frac{\partial d(\vec{x})}{\partial y}, \frac{\partial d(\vec{x})}{\partial z} \right), \tag{3.13}$$

and, following the upwind differencing algorithm 1, the proper derivative, forward ϕ_q^+ or backward ϕ_q^- for $q = x, y, z$, are chosen depending on the corresponding sign of $\frac{\partial d(\vec{x})}{\partial q}$. As forward and backward derivatives, different numerical approximations can be used, such as first-order derivatives or even the WENO scheme if more accuracy is required.

Regarding the second term, when ϕ is a SDF, the curvature κ can be approximated by (2.36). Additionally, central second-order derivatives must be used for solving this term. Because of this term, it is necessary to take a restrictive time step $\Delta t = O(\Delta x^2)$. In particular, if a forward first-order Euler time step is applied, (3.12) turns out:

$$\phi^{n+1} = \phi^n + \Delta t \left[\nabla d(\vec{x}) \nabla \phi^n + d(\vec{x}) \kappa^n |\nabla \phi^n| \right], \tag{3.14}$$

and the corresponding CFL condition (2.58) has to be used in order to avoid divergence, resulting in:

$$\Delta t < \frac{1}{\max \left\{ \frac{2d(\vec{x})}{(\Delta x)^2} + \frac{2d(\vec{x})}{(\Delta y)^2} + \frac{2d(\vec{x})}{(\Delta z)^2} \right\}}. \tag{3.15}$$

In particular, the following time step has been utilized to ensure convergence in all the tested cases:

$$\Delta t = \frac{0.9}{\max \left\{ \frac{2d(\vec{x})}{(\Delta x)^2} + \frac{2d(\vec{x})}{(\Delta y)^2} + \frac{2d(\vec{x})}{(\Delta z)^2} \right\}} \tag{3.16}$$

Notice that, although the first term only has a time step $\Delta t = O(\Delta x)$, the restriction of this second term applies to the whole equation since it is the most restrictive.

3.2.2 The convection model approach

Due to restrictive time step required by the energy model represented by the parabolic curvature-dependent equation (3.14), many iterations may be required in order to achieve the minimal energy surface.

Depending on the scenario, a simpler and faster model can result more interesting. Accordingly, the convection model was introduced [272]. In this model the surface velocity is defined by

$$\frac{\partial \Gamma(t)}{\partial t} = -\nabla d(\vec{x}). \tag{3.17}$$

Thus, the LS equation results in:

$$\phi_t - \nabla d(\vec{x}) \cdot \nabla \phi = 0. \tag{3.18}$$

In this model, each point of the surface is moved towards its closest S point until the whole surface reaches a local equilibrium, which corresponds with a linear approximation that connects all the points.

3.2.2.1 Numerical schemes

The simple hyperbolic LS equation (3.18) corresponds to a motion generated by a completely external field, as described in section 2.2.1, with $\vec{V} = -\nabla d(\vec{x})$. Accordingly, the upwind differencing method has to be used analogously as in the energy model to solve the first term of (3.14).

Regarding time discretization, if a forward first-order Euler time step is applied, (3.18) turns out:

$$\phi^{n+1} = \phi^n + \Delta t \left[\nabla d(\vec{x}) \cdot \nabla \phi^n\right] \tag{3.19}$$

and the CFL condition (2.45) result in:

$$\Delta t < \frac{1}{\max\left\{\frac{|D_x|}{\Delta x} + \frac{|D_y|}{\Delta y} + \frac{|D_z|}{\Delta z}\right\}}, \tag{3.20}$$

where $D_q = \frac{\partial d(\vec{x})}{\partial q}$, for $q = x, y, z$. In order to fulfil this CFL condition, the next time step has been employed:

$$\Delta t = \frac{0.5}{\max\left\{\frac{|D_x|}{\Delta x} + \frac{|D_y|}{\Delta y} + \frac{|D_z|}{\Delta z}\right\}}, \tag{3.21}$$

ensuring convergence in all the tested examples. The convection model can be solved with a time step $\Delta t = O(\Delta x)$ because of the curvature dependency has been removed, thus, less iterations than in the energy model (3.11) are required to reach the S points.

3.2.3 Mesh generation

The first step in both models (energy and convection) is to build a three-dimensional mesh enclosing all the S points. Since the CCA resulting structures follow the crystallographic structure of silicon, distances between atoms is known. Thus, if the substrate size of the structure is known as well as the unit lattice cells used in CCA simulations, the minimal distance between atoms *dist_min* can be obtained directly, avoiding to calculate the distances between each two points and selecting the lowest. Hence, the mesh resolution is proportional to this minimal distance between atoms:

$$\Delta x = n \cdot dist\_min, \tag{3.22}$$

where n is a positive and real number, which is chosen depending on the required accuracy of the reconstructed surface. A lower n will result in a finer grid with more points, which produces a more accurate surface. The effect of mesh resolution on reconstructed surface accuracy is studied in section 3.4.1. Furthermore, a regular mesh such that $\Delta x = \Delta y = \Delta z$ is built.

Once the resolution grid is chosen, the absolute dimensions must be specified. Accordingly, the minimal min_q and maximal max_q values in each dimension $q = x, y, z$, are determined among S points. Now, an offset value of $4\Delta x$ is added to each minimal and maximal value to allow the determination of a proper initial surface exterior to S points and surrounding them (see section 3.2.5). Then, the number of mesh points in each dimension is calculated, such that:

$$num_q = \left\lfloor \frac{max_q - min_q}{\Delta x} \right\rfloor. \tag{3.23}$$

The mesh generation process is summarized in algorithm 8.

Algorithm 8: Mesh generation algorithm

```
1 Determination of $dist\_min$ according to $S$ points features.
2 Selection of resolution $\Delta x$ according to the desired accuracy.
3 Determination of minimal and maximal values $min_q$ and $max_q$ for $q = x, y, z$.
4 Extension of the mesh dimensions by $min_q = min_q + 4\Delta x$ and
  $max_q = max_q + 4\Delta x$.
5 Calculation of number of mesh points with (3.23).
```

3.2.4 Distance matrix

Once the mesh has been generated, distance matrix $d(\vec{x})$ needs to be calculated. This matrix contains the distance between each mesh point and its closest S point. This task could result very costly if a direct algorithm is used since

$$num_x \cdot num_y \cdot num_z \cdot num_S, \tag{3.24}$$

distance calculations would be necessary (being num_S the number of points in S) and then, the minimal one for each grid point would have to be selected. In order to drastically reduce this computational effort, a propagating algorithm is proposed. The main idea is to discretize S points $\vec{x_S} = (x_S, y_S, z_S)$ over the mesh:

$$\begin{aligned} i &= \text{round}\left\{\frac{x_S - min_x}{\Delta x}\right\} \\ j &= \text{round}\left\{\frac{y_S - min_y}{\Delta x}\right\} \\ k &= \text{round}\left\{\frac{z_S - min_z}{\Delta x}\right\}. \end{aligned} \tag{3.25}$$

Then, for the corresponding mesh points (i, j, k) the exact distance to the closest $\vec{x_S}$ points is calculated. After that, these mesh points propagate the coordinates of their closest S points to their neighbouring mesh points so they can calculate the exact distance to the S points. This process is repeated until every mesh point has a distance value assigned. Accordingly, two *stacks* are used to track the current mesh points, thus, each mesh point is only visited six times (one per neighbouring point) and only six distance values are calculated per mesh point. The details of the proposed propagating algorithm are presented in algorithm 9.

Algorithm 9: Propagating algorithm for matrix distance calculation.

```
1  For every $\vec{x_S}$ point, determine the best mesh point $x_{i,j,k}$ approximation with
   (3.25). This $\vec{x_S}$ is associated to $x_{i,j,k}$ as its Closest $S$ Point (CSP). In case of two
   $\vec{x_S}$ are approximated to the same mesh point, only the closest will be associated
   as the CSP of the corresponding mesh point.
2  Assign an initial distance value to every mesh point, ensuring that this value is
   higher than all the possible distance values.
3  Calculate the exact distance between the $x_{i,j,k}$ points and their CSP obtained in
   step 1.
4  Initialize stacks $P_1$ and $P_2$.
5  Add all the $x_{i,j,k}$ points obtained in step 1 to $P_1$.
   while $P_1$ and $P_2$ are not empty do
       for every $p_1$ point from $P_1$ do
6          Calculate the distance between each of the six neighbouring points $np_1$ of
           $p_1$ and the CSP associated with $p_1$.
           if the new distance value is smaller than the previously stored then
7              Store the new one.
8              If $np_1$ is not included in $P_2$, add it.
       for every $p_2$ point from $P_2$ do
9          Calculate the distance between each of the six neighbouring points $np_2$ of
           $p_2$ and the CSP associated with $p_2$.
           if the new distance value is smaller than the previously stored then
10             Store the new one.
11             If $np_2$ is not included in $P_1$, add it.
```

This algorithm may produce a small error when calculating the matrix distance. For instance, if the mesh is not enough accurate, two S points can be discretized to the same mesh point even they are not at the same distance. Thus, only the closer of these two points will be considered in the propagating algorithm. Nevertheless, the distance matrices produced by this algorithm for several examples have been compared to the same matrices obtained by a direct algorithm (which ensures a perfect minimum distance calculation for every point), proving that the proposed propagating algorithm only produces a maximum error lower than mesh resolution (usually lower than $\Delta x/3$) and an average error per mesh point 4 or 5 orders of magnitude lower than mesh resolution. After comparing the resulting structures of both algorithms, it is concluded that the produced error is negligible.

Furthermore, both the energy (3.14) and the convection model (3.19) use the gradient of the matrix distance $\nabla d(\vec{x})$ (3.13), which is constant in the whole simulation and it is used by both models for the application of the upwind differencing scheme. To calculate the three matrices $\frac{\partial d(\vec{x})}{\partial q}$, for $q = x, y, z$, second-order central derivatives (2.28) are used in each spatial dimension.

3.2.5 Initial surface determination

Starting the evolution surface process with a proper initial surface is very important since ensures the convergence to S points as well as computational cost can be reduced. Instead of starting the reconstruction process with a trivial surface like a rectangular cuboid (Fig. 3.2 depicts an equivalent two-dimensional example) a contour of the distance matrix $d(\vec{x})$ is chosen as initial surface. In particular, for the MEMS structures studied in this thesis, the distance value chosen as contour is

$$\epsilon = 2\Delta x, \tag{3.26}$$

which ensures the convergence of the surface to S points with a few iterations while being far enough so the surface can reproduce properly the shape of the structure.

To determine those mesh points that satisfies condition (3.26) and encloses all the S points, an iterative tagging algorithm has been implemented based on the one presented by Zhao et al. [272]. The proposed procedure is presented in algorithm 10.

By applying this algorithm, all the exterior mesh points with a distance value lower than ϵ are tagged as initial surface. Additionally, the rest of exterior and interior mesh points, with respect to this new initial surface, are tagged accordingly.

Now, every mesh point is tagged as *interior*, *exterior* or *initial surface*. This enables to embed this initial surface inside an implicit SDF ϕ. First, distances between every mesh point and initial surface need to be determined. To accomplish this task, the propagating algorithm 9 is used. However, instead of discretizing

Algorithm 10: Determination of initial surface.

```
1 Tag every mesh point as *interior* but the two most exterior rectangular cuboids
  that are tagged as *exterior*.
2 Initialize stack $P$ and include every *exterior* point.
  while $P$ is not empty do
3     Extract the first $p$ point from $P$.
      for each neighbouring point $np$ of $p$ do
          if $np$ is interior then
              if distance value assigned to $np \geq \epsilon$ then
4                 Tag $np$ as *exterior*.
5                 Add $np$ to stack $P$.
              else
6                 Tag $np$ as *initial surface*.
```

the $\vec{x_S}$ points, the mesh points tagged as *initial surface* will be used, such that they are their own CSP. This produces a zero distance value that corresponds to the zero LS of ϕ.

Once every mesh point has a distance value to its closest *initial surface* point, these values are signed according to their tags, i.e. negative for *interior* and positive for *exterior* points. Therefore, the SDF ϕ is built.

Notice that local LS methods like the SFM do not require ϕ values for all the mesh points, but only for those active points close to the surface. Hence, a more efficient algorithm is explained in section 3.3.2 instead of using the propagating algorithm.

3.3 Developed implementations

When implicit SDF ϕ has been obtained, the evolution process of the surface can be started. In this thesis two LS-based implementations for image reconstruction have been implemented for comparison. In the first one, the whole grid space is computed, whereas in the second, a local SFM implementation has been developed.

In both implementations, once the evolution surface process is finished, the final step is to extract the implicit surface from the corresponding SDF for visualization. The MATLAB function *isosurface* is used for this task. This function takes the mesh, the ϕ values and the contour value (the zero level) as input and it produces a list of explicit faces and vertices that can be visualized.

3.3.1 Original LS

As commented in section 2.1.6, the implicit function ϕ must be maintained as a SDF during the whole front evolution process in order to guarantee convergence, accurate results, and the application of SDF properties (see section 2.1.5). In this first implementation, the reinitialization method presented by Sussman et al. has been used [42].

This technique keeps the implicit function ϕ as a SDF by taking

$$\phi_t = Sgn(\phi_0)\left(1 - |\nabla\phi|\right) \tag{3.27}$$

to steady state after each LS iteration, where Sgn is the smoothed sign function

$$Sgn(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + \Delta x^2}}, \tag{3.28}$$

and ϕ_0 is the SDF of initial surface. If a simple Euler time step is applied to (3.27), it turns out:

$$\frac{\tilde{\phi} - \phi}{\Delta\tilde{t}} = Sgn(\phi_0)(1 - |\nabla\phi|), \tag{3.29}$$

where $\Delta\tilde{t}$ is the time step applied in the reinitialization process, ϕ is the implicit function resulting from LS iterations and $\tilde{\phi}$ is the new SDF that can be obtained with:

$$\tilde{\phi} = \phi + \Delta\tilde{t} \cdot Sgn(\phi_0)(1 - |\nabla\phi|). \tag{3.30}$$

A typical and proper value of the time step is

$$\Delta\tilde{t} = \Delta x/10, \tag{3.31}$$

and the operator $|\nabla\phi_{i,j,k}|$, for a generic grid point i, j, k, must be calculated with the next expression to ensure stability [42]:

$$|\nabla\phi_{i,j,k}| = \begin{cases} \sqrt{\max\left\{A_x^t, B_x^b\right\} + \max\left\{A_y^t, B_y^b\right\} + \max\left\{A_z^t, B_z^b\right\}} & \text{if } \phi_{i,j,k}^0 > 0 \\ \sqrt{\max\left\{A_x^b, B_x^t\right\} + \max\left\{A_y^b, B_y^t\right\} + \max\left\{A_z^b, B_z^t\right\}} & \text{if } \phi_{i,j,k}^0 < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.32}$$

where

$$\begin{aligned} A_q^t &= \left(\max\left\{\phi_p^-, 0\right\}\right)^2, \quad A_q^b = \left(\min\left\{\phi_p^-, 0\right\}\right)^2 \\ B_q^t &= \left(\max\left\{\phi_p^+, 0\right\}\right)^2, \quad B_q^b = \left(\min\left\{\phi_p^+, 0\right\}\right)^2, \end{aligned} \tag{3.33}$$

being $q = x, y$ or z such that ϕ_q^- is the backward derivative in dimension q and ϕ_q^+ is the analogous forward derivative. Notice that A and B are chosen to represent

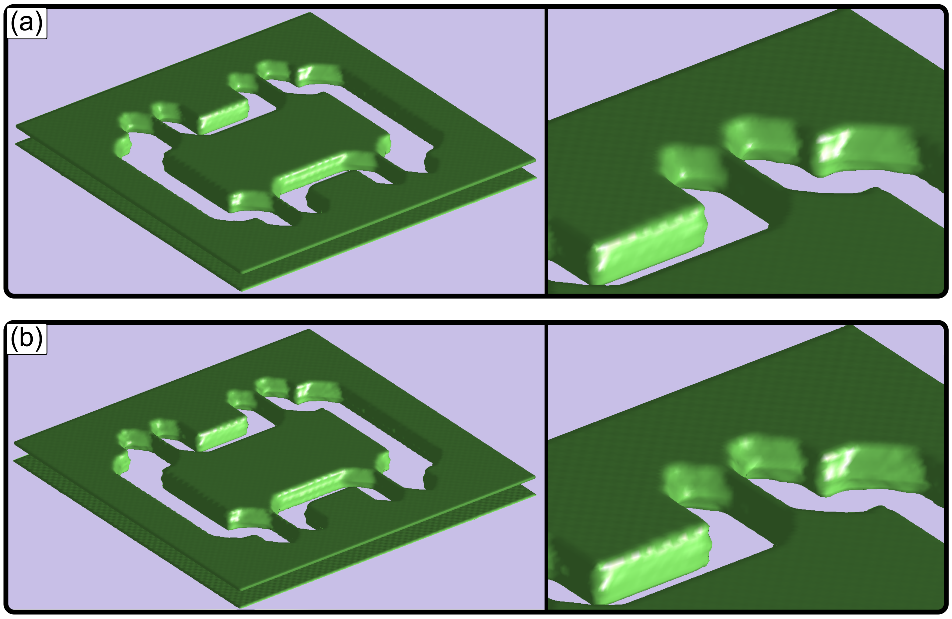


Figure 3.3: Comparison of original LS implementation using: (a) first-order accurate spatial differences and (b) fifth-order WENO scheme.

backward and forward differences, whereas the superscripts t and b indicate the max or min operator respectively.

By applying this method, the zero level of ϕ is kept intact but the rest of the levels are modified to fulfil the properties of a SDF [42].

Due to the high computational effort of computing the whole three dimensional grid, the two models are used to accelerate the executing process, i.e. the convection model is used in first place for a few iterations and then the surface is adapted to S points with the energy model. In particular, for an initial surface obtained like commented in section 3.2.5, only 3 iterations of the convection model are sufficient to accelerate the total simulation time significantly while allowing the energy model to correctly adapt.

In this thesis, two different approaches for spatial backward and forward derivatives have been implemented, namely, the fifth-order accurate WENO scheme [53, 54] and the first-order simple derivatives (2.27) and (2.26). Nevertheless, due to the higher computational effort required by WENO scheme, the same simulation process takes about 3 to 5 times more than first-order differences. Fig. 3.3 shows a comparison of a surface reconstructed by both schemes when they are applied with the same conditions and for the same number of iterations. For comparison, an accelerometer structure has been chosen [142]. As can be observed, although the result of first-order derivatives looks softer and less accurate, both are very similar. Thus, the first-order accurate difference approach has been selected.

The pseudocode of this original LS-based implementation is presented in algorithm 11. Notice that the variables $\phi, \tilde{\phi}, \nabla\phi, |\nabla\phi|, d(\vec{x}), D_q, \phi_q, \phi_q^{\pm}, \kappa$ are three-dimensional variables of mesh size, storing the corresponding value for every mesh point. Hence, all the operations are performed for each point individually. For instance, when applying the upwind differencing, the proper derivative $\phi_x(i,j,k)$ of the point (i,j,k) is selected by checking the sign of $D_x(i,j,k)$, which can be different for another point $(i2,j2,k2)$. Nevertheless, for simplicity and clarity, in algorithm 11 the individual-point notation is omitted.

3.3.2 Local SFM

Another LS implementation for evolving the initial surface up to S points has been implemented. This second approach implements the local SFM, thus, after each time step a perfect SDF is built, avoiding to use any reinitialization process.

Similarly to the previous implementation, only first-order derivatives are used, hence, according to the SFM, the lists L_0, L_{+1}, and L_{-1} as well as the temporary lists S_0, S_{+1}, and S_{-1} are used to keep the active points updated (see section 2.2.4.2). Additionally, each grid point is labelled with the state $0, 1, -1, 2, -2$ depending on their ϕ value. This information is collected in table 2.1.

Accordingly, the implicit ϕ function is only needed at those points close to the surface. Thus, there is no need to use the propagating algorithm 9 to build ϕ over the whole mesh. Instead of this and taking into account the point tags (or states) produced by initial surface determination algorithm 10 (i.e. *exterior*, *interior* or *initial surface (IS)*), the algorithm 12 is used to embed the surface in the SDF ϕ.

This implementation also applies, in first place, three iterations of the convection model, followed by the energy model until convergence is achieved. According to procedures 4, 5, 6, and 7, the steps of this implementation are shown in algorithm 13.

Algorithm 11: Original LS image reconstruction implementation.

```
1  Generate a three-dimensional mesh according to algorithm 8.
2  Build the distance matrix $d(\vec{x})$ between mesh and $S$ points with the propagating
   algorithm 9.
3  Calculate $\nabla d(\vec{x})$ with second-order central derivatives (2.28).
4  Find a proper initial surface with algorithm 10.
5  Build the SDF $\phi$ of the initial surface by applying the propagating algorithm 9
   and keep these values ($\phi_0$) for reinitialization process.
6  Calculate the time steps for convection model $\Delta t_c$, energy model $\Delta t_e$, and
   reinitialization process $\Delta\tilde{t}$, according to (3.21), (3.16), and (3.31) respectively.

                                                                -Convection model-
   for iter=0; iter<3; i++ do
7      Calculate forward and backward spatial derivatives $\phi_x^+, \phi_x^-, \phi_y^+, \phi_y^-, \phi_z^+, \phi_z^-$.
8      Use $D_q$ to determine by upwind differencing algorithm 1 the proper derivatives,
       backward of forward, in each dimension $q$.
9      Update implicit function $\phi$ with (3.19) using $\Delta t_c$.
10     Calculates $|\nabla\phi|$ according to (3.32).
11     Apply the reinitialization process to $\phi$ by (3.30) to obtain $\tilde{\phi}$.
12     Do $\phi = \tilde{\phi}$.
                                                                    -Energy model-
   while no convergence do
13     Calculate $|\nabla\phi|$ with second-order central derivatives (2.28).
14     Approximate curvature $\kappa$ with (2.36).
15     Calculate forward and backward spatial derivatives $\phi_x^+, \phi_x^-, \phi_y^+, \phi_y^-, \phi_z^+, \phi_z^-$.
16     Use $D_q$ to determine by upwind differencing algorithm 1 the proper derivatives,
       backward of forward, in each dimension $q$.
17     Update implicit function $\phi$ with (3.14) using $\Delta t_e$.
18     Calculates $|\nabla\phi|$ according to (3.32).
19     Apply the reinitialization process to $\phi$ by (3.30) to obtain $\tilde{\phi}$.
20     Do $\phi = \tilde{\phi}$.
21 Extract implicit final surface for visualization.
```

Algorithm 12: Determination of implicit SDF ϕ in the SFM implementation.

```
Data: $state = \{exterior,\ interior,\ IS\}$
Data: List of active points $= \{L_{+1}, L_{-1}, L_0\}$
for each mesh point $(i, j, k)$ do
    switch $state$ do
        case $IS$
1           $\phi(i, j, k) = 0$
2           $state\_aux(i, j, k) = 0$
3           add $(i, j, k)$ to $L_0$
4           break;
        case $exterior$
            if any adjacent point of $(i, j, k)$ has state $= IS$ then
5               $\phi(i, j, k) = \Delta x$
6               $state\_aux(i, j, k) = 1$
7               add $(i, j, k)$ to $L_{+1}$
            else
8               $\phi(i, j, k) = 1.5\Delta x$
9               $state\_aux(i, j, k) = 2$
10          break;
        case $interior$
            if any adjacent point of $(i, j, k)$ has state $= IS$ then
11              $\phi(i, j, k) = -\Delta x$
12              $state\_aux(i, j, k) = -1$
13              add $(i, j, k)$ to $L_{-1}$
            else
14              $\phi(i, j, k) = -1.5\Delta x$
15              $state\_aux(i, j, k) = -2$
16          break;
17 $state = state\_aux$
```

Algorithm 13: SFM implementation for surface reconstruction.

```
1  Generate a three-dimensional mesh according to procedure 8.
2  Build the distance matrix $d(\vec{x})$ between mesh and $S$ points with the propagating
   algorithm 9.
3  Calculate $\nabla d(\vec{x})$ with second-order central derivatives (2.28).
4  Find a proper initial surface with algorithm 10.
5  Build the SDF $\phi$ of the initial surface and add the points to the corresponding lists with
   algorithm 12 .
6  Calculate the time steps of the convection model $\Delta t_c$ and the energy model $\Delta t_e$ according
   to (3.21) and (3.16) respectively.
                                                                    -Convection model-
   for iter=0; iter<3; i++ do
       for each $L_0$ point $\vec{x}_i$ do
7          Calculate $\phi_x^+, \phi_x^-, \phi_y^+, \phi_y^-, \phi_z^+, \phi_z^-$.
8          Use $D_q(\vec{x}_i)$ to determine, by upwind differencing algorithm 1, the proper
           derivative to use in each dimension $q$.
9          Update implicit function $\phi$ with (3.19) using $\Delta t_c$.
           if $\phi(\vec{x}_i) < -0.5\Delta x$ then
10             remove $\vec{x}_i$ from $L_0$ and add it to $S_{-1}$.
           if $\phi(\vec{x}_i) > 0.5\Delta x$ then
11             remove $\vec{x}_i$ from $L_0$ and add it to $S_{+1}$.
12     Update $L_{+1}$ and $L_{-1}$ lists with procedure 5.
13     Transfer points from auxiliary lists by applying procedure 6.
14     Add corresponding points to $L_{\pm 1}$ according to procedure 7.
                                                                    -Energy model-
   while no convergence do
       for each $L_0$ point $\vec{x}_i$ do
15         Calculate $\phi_x^+, \phi_x^-, \phi_y^+, \phi_y^-, \phi_z^+, \phi_z^-$.
16         Use $D_q(\vec{x}_i)$ to determine by upwind differencing algorithm 1 the proper derivative
           to use in each dimension $q$.
17         Calculate $|\nabla \phi|$ with second-order central derivatives (2.28).
18         Approximate curvature $\kappa$ with (2.36).
19         Update implicit function $\phi$ with (3.14) using $\Delta t_e$.
           if $\phi(\vec{x}_i) < -0.5\Delta x$ then
20             remove $\vec{x}_i$ from $L_0$ and add it to $S_{-1}$.
           if $\phi(\vec{x}_i) > 0.5\Delta x$ then
21             remove $\vec{x}_i$ from $L_0$ and add it to $S_{+1}$.
22     Update $L_{+1}$ and $L_{-1}$ lists with procedure 5.
23     Transfer points from auxiliary lists by applying procedure 6.
24     Add corresponding points to $L_{\pm 1}$ according to procedure 7.
25 Extract implicit final surface for visualization.
```

3.4 Results

In this section, first, the effect of the mesh resolution used by the LS method on the reconstructed surfaces is studied. Later, several surfaces reconstructed with both the original LS-based (algorithm 11) and the local SFM-based (algorithm 13) implementations are shown. Every reconstructed surface is compared with the corresponding CCA result in order to visualize the improvement obtained with the LS implementations. In addition, some features of resulting structures are measured in order to calculate the error produced by both implementations. CCA results are obtained with Intellietch [197] which applies a colouring technique to atoms regarding their normal vectors. Finally, a discussion about the execution times of the implementations is presented.

A compilation of all the common simulation parameters and numerical techniques used in both implementation are collected in table 3.1. Both implementations have been written using sequential Java programming language. Moreover, they are executed on a testing machine consisting of Intel core i7 at 3.4 GHz with 8 GB of RAM using 64 bit Windows-based server and Java Virtual Machine (version 1.8.0_31).

It is important to notice that CCA structures results are formed only by surface atoms, therefore, the most of the times they are not closed surfaces. However, the LS implementations, start from an initial closed surface that shrinks up to S points. For this reason, a set of points can be reached by two different LS surfaces (for example from the top and the bottom) at the same time, producing holes in the surface. Thus, the energy of the surface is not always able to achieve the local minimum that corresponds with the final surface (contrary to the developed LS application of chapter 5, which does allow to achieve a local minimum of energy because of the type of surfaces).

Accordingly, it has been found a proper amount of iterations that, given the initial surface obtained according to section 3.2.5, produces accurate and closed surfaces, improving significantly results visualization. Particularly, in the original LS implementation, 3 iterations with the convection model, followed by 40 of the energy model. On the other hand, in the SFM a perfect SDF is built after each iteration and the SDF is not modified by the reinitialization process, therefore between 40 and 70 iterations of the energy model are required in this implementation. Similarly, 3 iterations of the convection model are previously applied.

Expression	Description
$\Delta x = n \cdot dist\_min$	Mesh resolution proportional to the minimal distance between CCA atoms.
$\Delta t_c = \frac{0.5\Delta x}{\max\{\lvert D_x\rvert+\lvert D_y\rvert+\lvert D_z\rvert\}}$	Time step for convection model.
$\Delta t_e = \frac{0.9\Delta x^2}{6\max\{d(\vec{x})\}}$	Time step for energy model.
$\Delta\tilde{t} = \frac{\Delta x}{10}$	Time step for reinitialization process (only used in original LS implementation).
$\nabla d(\vec{x}) = (D_x, D_y, D_z)$	Gradient of distance matrix.
$D_q = \frac{\phi_{l+1}-\phi_{l-1}}{2\Delta}$	Spatial second-order central derivatives of distance matrix for $q = x, y, z$ and $l = i, j, k$.
$\phi_q^+ = \frac{\phi_{l+1}-\phi_l}{\Delta x}$	Forward first-order derivative of ϕ.
$\phi_q^- = \frac{\phi_l-\phi_{l-1}}{\Delta x}$	Backward first-order derivative of ϕ.
$\kappa = \phi_{xx}+\phi_{yy}+\phi_{zz}$	Approximation of the local curvature for a SDF.
$\phi_{qq} = \frac{\phi_{l+1}-2\phi_l+\phi_{l-1}}{\Delta x^2}$	Second-order accurate approximation for the second derivatives of $q = x, y, z$.

Table 3.1: Numerical techniques and values used in both LS and SFM developed implementations (algorithms 11 and 13 respectively).

3.4.1 Impact of the mesh resolution

The resolution of the mesh used by the LS method has an important impact on the accuracy of the reconstructed surface. In order to study this effect, the accelerometer of Fig. 3.3 has been reconstructed with several resolution values using both implementations. Furthermore, two distances of the structure are measured to compare the error with the original CCA result. This CCA structure can be visualized in Fig. 3.4, including the mask used in the wet etching process and a close-up, and it has the following features:

- Silicon substrate size: 1260x1260x70 μm^3.
- Surface orientation: (100).
- Number of UCs: 128x128.

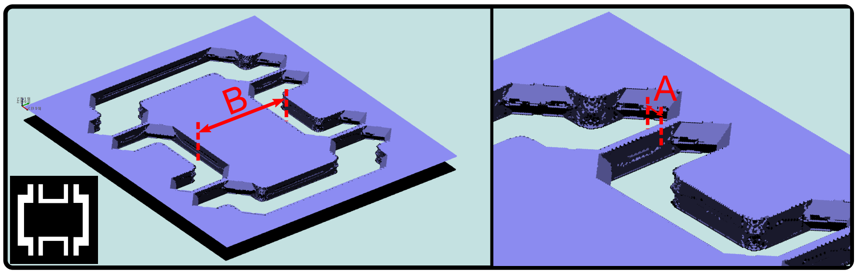


Figure 3.4: CCA simulation result of a simple accelerometer micromachining process. Measurements A and B are shown for comparison with LS reconstructions.

- Measurement A (width of the beam): 14.8 μm.
- Measurement B (width of the central mass): 419.6 μm.

Seven different resolutions have been used to build the LS mesh in order to reconstruct this cloud of points. These resolutions have been chosen such that each one has the double of mesh points than the previous resolution approximately, since the margin points added in each dimension are not considered (see section 3.2.3). Since it is a three-dimensional grid, each dimension must be multiplied by $\sqrt[3]{2}$. For each reconstruction process, table 3.2 shows: the value n which multiplies *dist_min* to produce Δx (column 1), the corresponding mesh dimensions in voxels [1](column 2), the iterations of the energy model (the convection model is always applied for 3 iterations) (column 3), the execution time of the whole process (column 4), the distances of measurements A and B of the reconstructed structures (columns 5 and 6), and the relative errors of these measurements (columns 7 and 8), i.e. the difference between CCA and reconstructed measurements divided by the mesh resolution. Each row is subdivided into two values in order to present results of both implementations, the upper values are the original LS results and the lowers correspond to the SFM implementation ones.

As can be observed in table 3.2, in the most of the cases the errors are smaller than mesh resolution. Although a finer resolution does not imply a smaller error (in a specific measurement), the accuracy of reconstructed surfaces is actually increased since a more detailed visualization of planes is observed as shows Fig. 3.5. This figure shows the reconstructed surfaces corresponding to several cases of table 3.2, including measurements A and B.

[1]The term voxel refers to volumetric pixel, and it is the minimal cubic unit that forms a three-dimensional object. In the LS method, each mesh point has a distance value assigned that defines the embedded surface and, thus, every mesh point corresponds to a voxel.

$\frac{\Delta x}{dist\_min}$	**Mesh (voxels)**	**Energy iter.**	**Recons. time (s)**	**Measure A (μm)**	**Measure B (μm)**	$\frac{\textbf{error A}}{\Delta x}$	$\frac{\textbf{error B}}{\Delta x}$
1.0	598x598x39	40	94.0	16.6	420.3	0.86	0.33
		70	37.0	16.9	421.8	1.00	1.03
1.26	476x476x33	40	50.8	17.5	422.9	1.016	1.23
		70	24.6	17.4	421.9	0.98	0.85
1.5876	380x380x28	40	27.9	16.4	422.3	0.48	0.80
		70	13.3	16.0	421.6	0.36	0.60
2.0	303x303x24	40	15.6	18.0	421.5	0.76	0.45
		70	7.3	17.7	421.9	0.69	0.35
2.5204	242x242x21	40	10.1	17.9	425.7	0.58	1.13
		50	4.1	18.7	425.5	0.73	1.10
3.18	194x194x18	40	7.2	20.2	426.4	0.8	1.00
		50	3.3	20.6	426.1	0.86	0.96
4.0	156x156x16	40	4.6	17.4	420.8	0.3	0.14
		40	1.6	19.8	423.6	0.59	0.47

Table 3.2: Reconstruction parameters and results, including measurements of the simple accelerometer used for visualizing the effect of mesh resolution. For each case, both the original LS (upper rows) and the SFM implementation (lower rows) results are presented.

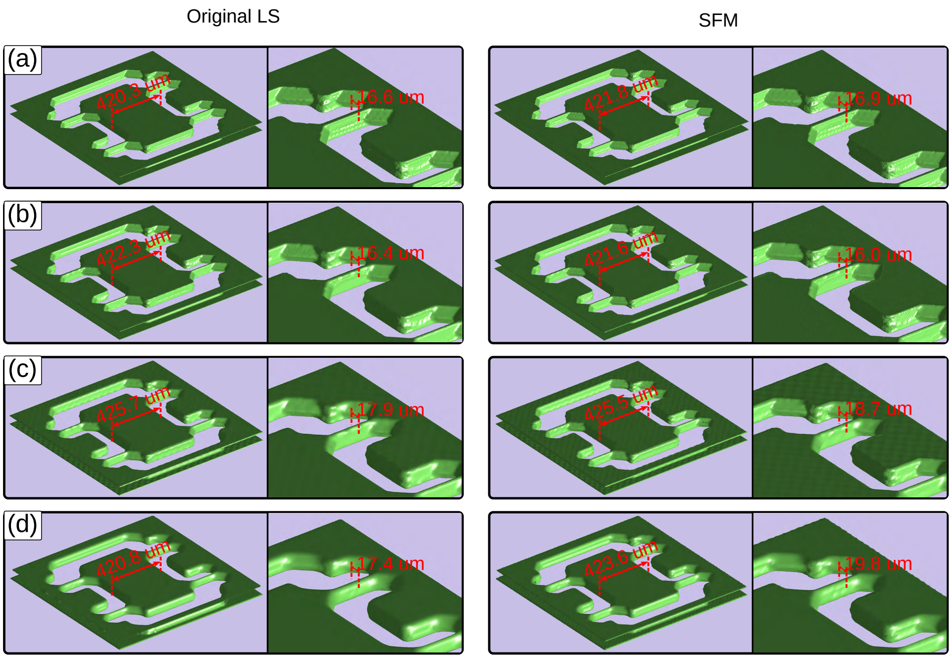


Figure 3.5: Representation of the effect of mesh resolution in the original LS and the SFM implementations, by reconstructing a simple accelerometer with four different resolutions: $\frac{\Delta x}{dist\_min}$ = (a) 1.0, (b) 1.5876, (c) 2.5204, and (d) 4.0. Measurements A and B are shown in each case. The SFM presents sharper results than original LS implementation.

Regarding the accuracy of the original LS in comparison with the SFM implementation, both methods present similar errors in all the cases and are in close proximity to each other. Nevertheless, the SFM presents sharper results, which is more stressed in higher resolutions (coarser meshes) (see Fig. 3.5).

In relation to execution times, in all the experiments the SFM has been more than twice faster than original LS while obtaining better results, proving the efficacy and efficiency of this technique. Moreover, the relation of computational cost with mesh size can be observed in Fig. 3.6. The original LS method computational cost is $O(N^3)$, being N the number of points in each dimension. For example, if the number of points in each dimension is doubled, i.e. the total mesh points is multiplied by eight, the execution time will be theoretically, eight times higher. Thus, execution time is directly proportional to total mesh points. This behaviour is observed in the left graph of Fig. 3.6, for example, for $\frac{\Delta x}{dist\_min} = 1.0$ the number of mesh points is 13946556, while for $\frac{\Delta x}{dist\_min} = 2.0$ the number of mesh points is 2203416, which is 6.33 times lower. Accordingly, the execution times

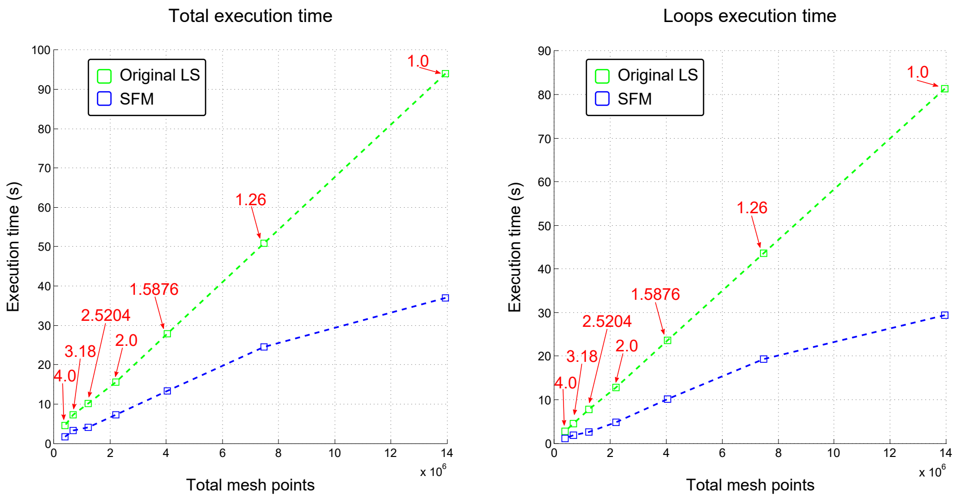


Figure 3.6: Representation of the computational time of both implementations for the mesh sizes of table 3.2. In red are shown the corresponding $\frac{\Delta x}{dist\_min}$ factors. The left graph represents the execution time of the whole algorithms whereas in the right graph only the loops evolution (convection and energy) times are considered.

are 94.0 and 15.6 s respectively, i.e. a relation of 6.03 that is in close proximity to the theoretical value. However, this behaviour is not completely achieved in coarser meshes since other parts of the algorithm, like matrix distance calculation, become more relevant (21.7% of the total execution time for $\frac{\Delta x}{dist\_min} = 4.0$ in contrast with 9.4% for $\frac{\Delta x}{dist\_min} = 1.0$). If only the execution time of the evolution loops (convection and energy) are considered, this behaviour is almost perfectly achieved, as shown in the right graph of Fig. 3.6, where the execution time of the loops follows an almost perfect straight line.

Similarly, the computational times of the SFM implementation are also shown in Fig. 3.6. The theoretical computational cost of the SFM is $O(N^2)$, so if the number of points in each dimension is doubled, i.e. the total mesh points is increased by a factor of four, the execution time is also multiplied by four. Thus, computational time is proportional to total mesh points as well as in the original LS but the proportion is lower. This behaviour is perfectly visualized in both graphs of Fig. 3.6. Accordingly, the case with $num_x = 598$ ($n = 1.0$) has an execution time of 37.0 s, while for the case $num_x = 303$ ($n = 2.0$) the execution time is 7.3 s which is in close proximity to the expected theoretical value 9.5 s. However, this theoretical behaviour is not perfectly achieved because of the strong dependency of the SFM on the surface morphology, i.e. this method not only depends on the mesh size but on the active points of the surface. Nevertheless, an important reduction of computational time is obtained by using the SFM-based implementation as show both graphs of Fig. 3.6.

Finally, according to the results of this section, in the next reconstruction processes examples, the mesh resolution

$$\Delta x = 1.5876 \cdot dist\_min, \tag{3.34}$$

will be used since it allows to obtain accurate results with a low error and preserves similar visualization details than $\Delta x = 1.0 \cdot dist\_min$, while keeping an execution time of a few seconds. Accordingly, the corresponding number of iterations will be used as well, i.e. 3 iterations of the convection model followed by 40 iterations of the energy model in the original LS implementation and 70 in the local SFM implementation.

3.4.2 Convex corners

The first example is a simple wet etching experiment of silicon that shows the underetching phenomenon and the formation of different crystallographic planes [274]. The details of the simulated experiment are:

- Silicon wafer (100) oriented with 2500x2500 μm^2 size.
- 128x128 UCs are used in the simulator.
- Micromachining process:
  - Application of the square mask pattern of silicon nitride shown in Fig. 3.7.
  - Wet etching with KOH 40 wt% at 81 °C for 190 minutes.
  - Remove the mask.
- Measurement A (width of the mesa): 1597 μm.
- Measurement B (long of the mesa): 1602 μm.
- Measurement C (height): 193.4 μm.

The simulated CCA result is shown in Fig. 3.7(a). Intellietch simulation tool applies a shading process to the CCA resulting points according to normal vector of discrete points. Nevertheless, in the close-up of the figure it is observed that the resulting structure is still a cloud of unconnected points, making the visualization of crystallographic planes difficult.

This surface is reconstructed using both implementations. These results are presented in Fig. 3.7(b) and (c) for the original LS and the SFM implementations

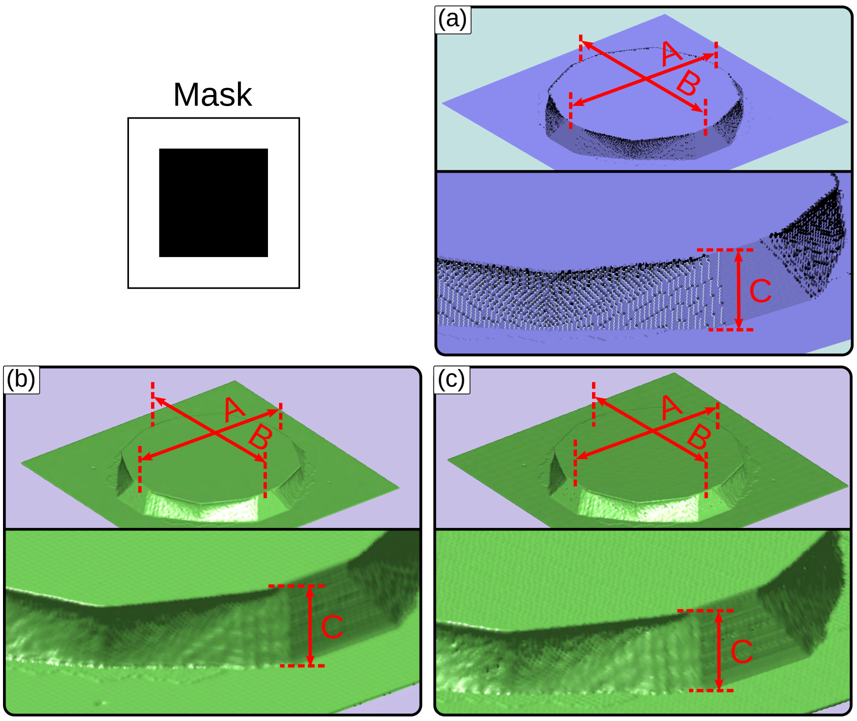


Figure 3.7: Convex corners results: (a) CCA, (b) original LS, and (c) SFM implementations. The mask used in the micromachining process is also shown.

respectively. Measurements A, B, and C are compiled in table 3.3 as well as the corresponding errors with respect to CCA measurements. Although both implementations present very similar results, the SFM errors are lower and the *steps* produced in the silicon surface are better observed (see close-ups of Fig. 3.7) in this implementation result. Additionally, it has been 2.6 times faster than original implementation.

3.4.3 Microneedles

The next example is an array of microneedles formed by a succession of dry and wet etching processes [275]:

- Silicon (100) wafer of 600x600 μm^2.
- 184x184 UCs.
- Micromachining process:

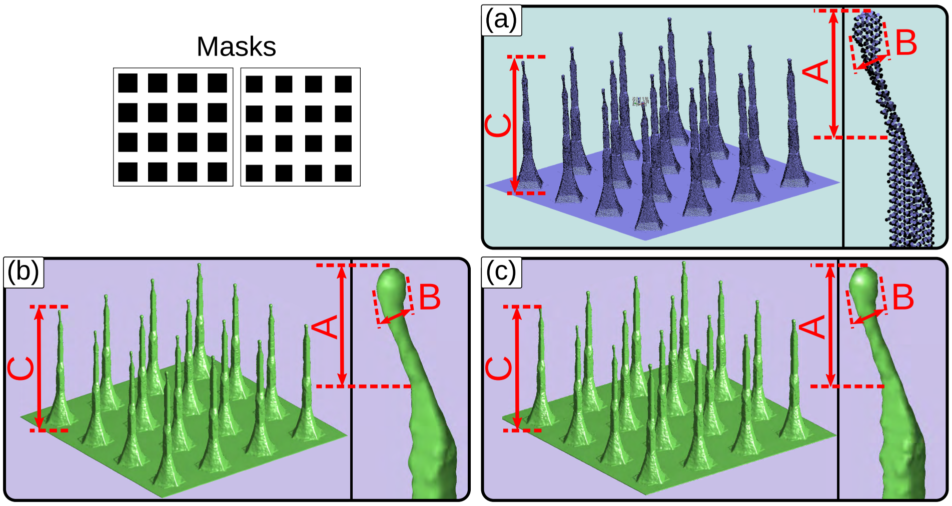


Figure 3.8: Microneedles results: (a) CCA, (b) original LS, and (c) SFM implementations. The masks used in the micromachining process are also shown.

  - Deposition of the left mask shown in Fig. 3.8.
  - DRIE up to 240 μm depth.
  - Remove the applied mask.
  - Apply the mask on the right.
  - DRIE up to 320 μm depth (additional 80 μm).
  - Remove the mask.
  - Wet etching with KOH 40 wt% at 70 °C.

- Measurement A (height of the top part of the needle): 36.3 μm.
- Measurement B (width of a needle head): 7.54 μm.
- Measurement C (height of a needle): 313.6 μm.

For this example a coarser mesh had to be used since, due to the high depth of the structure, many mesh points were created and too much memory was required. As a consequence, the testing machine was not able to reconstruct the structure. Thus, instead of using $\Delta x = 1.5876 \cdot dist\_min$, a value of $\Delta x = 2.274 \cdot dist\_min$ has been used. Nevertheless, the errors obtained by both implementations are still small.

In this example, the difference between execution times of original and SFM has been increased. Particularly, the SFM implementation has been 5.7 times faster as presented in table 3.3. This is because the majority of the mesh points are not computed by the SFM due to this particular structure. On the other hand, both results shown in Fig. 3.8(b) and (c) respectively, are in close proximity with each other and with the CCA result (a). The improvement obtained by both LS approaches can be especially well visualized in the close-up.

3.4.4 Accelerometer

Another reconstructed surface is the result of the micromachining process of a three-axis accelerometer [142]. Notice that, in this simulated experiment, the silicon wafer is etched at both sides (top and bottom):

- Silicon sample (100) with 3780x1260x70 μm^3 size.
- 512x170 UCs. Notice that only the surface atoms are processed by the CCA simulator Intellietch, thus, the vertical UCs dimension depends on the etching time and atoms are added during the simulation.
- Micromachining process:
  - Deposition of the upper mask shown in Fig. 3.9 on both sides of the wafer, top and bottom.
  - Wet etching with KOH 40 wt% at 70 °C for 20 min.
  - Remove the pattern (lower mask) of Fig. 3.9 from the top and the bottom sides.
  - Wet etching with KOH 40 wt% at 70 °C for 45 min.
  - Remove both masks.
- Measurement A (width of the first mass): 543.7 μm.
- Measurement B (width of the beam of the second module): 9.3 μm.
- Measurement C (width of the beam of the third module): 9.3 μm.

The resulting CCA structure is shown in Fig. 3.9(a). In this complex structure there are many different crystallographic orientations, which make difficult to calculate the normal vector of the atoms at those complicated topologies. A consequence of this can be visualized in the close-ups of Fig. 3.9(a), where many atoms are not properly coloured by the simulator.

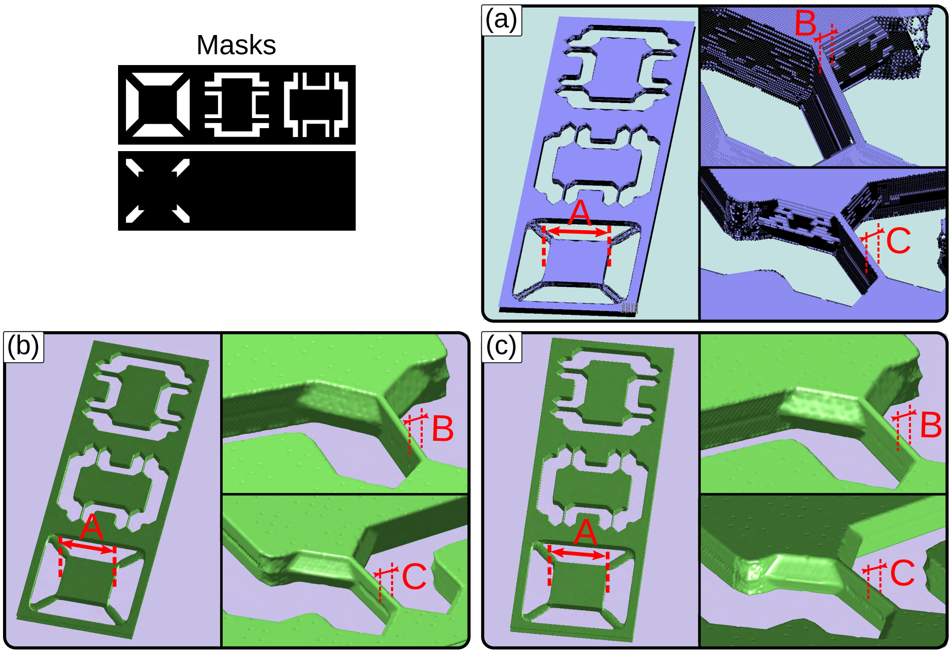


Figure 3.9: Accelerometer results: (a) CCA, (b) original LS, and (c) SFM implementations. The mask (upper) and the pattern (lower) used in the micromachining process are also shown.

In addition to these complicated topologies, this structure is formed by holes and accurate small parts, like the beams. Despite all of these difficulties, both LS implementations are capable to reconstruct flawlessly the surface as shown in Fig. 3.9(b) and (c). Especially interesting are the close-ups, showing those parts where the CCA presents a low accuracy but, anyhow the LS approaches could reconstruct the surface. In this case, both implementations achieved very similar results and errors (see table 3.3) but, again, the SFM is 2.9 times faster than original LS.

3.4.5 Wagon wheel

Finally a complex wagon wheel structure is used to prove the versatility of the developed LS-based implementations. Wagon wheels experiments are used to characterize a specific wet etching process [191]. The particular simulated experiment is:

- 200x200 μm^2 silicon wafer (100) oriented.
- 184x184 UCs.

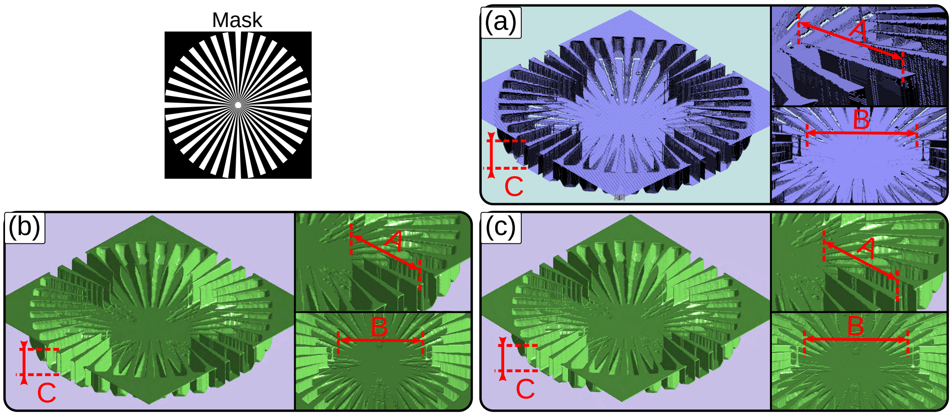


Figure 3.10: Wagon wheel results: (a) CCA, (b) original LS, and (c) SFM. The mask used in the micromachining process is also shown.

- Micromachining process:
  - Deposition of the mask shown in Fig. 3.10.
  - DRIE up to 20 μm of depth.
  - Wet etching with KOH 40 wt% at 70 °C for 3 min.
  - Remove the mask.
- Measurement A (long of the spoke): 55.8 μm.
- Measurement B (distance between the two central spokes): 89.13 μm.
- Measurement C (depth): 21.88 μm.

The reason these kind of experiments is used to obtain the etch rates of specific crystallographic orientations is because many complex planes appear during wet etching process. These planes correspond with the spokes of the resulting structure. As can be seen in the CCA result in Fig. 3.10(a), the spokes are very thin in some parts and, as a consequence, many atoms are not correctly represented by the CCA simulator, which can make hard to study this type of experiments (see close-ups of Fig. 3.10).

The results provided by both LS implementations present a significant improvement as can be seen in Fig. 3.10(b) and (c). Nevertheless, in this case the SFM presents more accurate results and lower errors, as exposed in table 3.3. In addition, the SFM implementation has been able to reconstruct even those thin parts of the spokes formed by single atoms, like the spokes used in measurements A and B (close-ups of Fig. 3.10(c)). Contrary, the original LS implementation has not been able to be so accurate and separated surfaces have been formed as can be observed in Fig. 3.10(b). Regarding the computational time, the SFM is 2.8 times faster.

3.4.6 Computational behaviour

Finally, Fig. 3.11 shows two graphs comparing the execution time of the mains parts of both implementations. Regarding the original LS implementation, the most costly part is clearly the evolution loops, which include the calculation of spatial derivatives and the update of SDF, and the application of the reinitialization process. These two parts of the loops cost practically the same in all the examples, taking, each one, more than 40% of the total time. Because of this, the matrix distance determination (which uses the propagating algorithm 9) only takes around 10% of the whole reconstruction process.

On the other hand, in the SFM implementation, especially the lists update subroutine takes significantly less time (between 15 and 30%) than the reinitialization process in the first implementation. Also, the time taken by this subroutine is lower than the calculation of spatial derivatives and the ϕ update. As a consequence, the propagating algorithm used in distance matrix determination, becomes more relevant, taking up to 47.6% of the time in the microneedles example.

Another interesting behaviour that can be seen in Fig. 3.11 is the reduction of the *build* ϕ subroutine contribution in the SFM implementation. This is because the propagating algorithm is avoid and the faster and direct solution of algorithm 12 is used.

A less important time-consuming part of the algorithm is labelled as *rest of algorithm* in Fig. 3.11. This part includes several steps like reading S points, initial surface determination, calculation of the time steps, mesh generation, and calculation of the derivatives of the matrix distances. All these tasks have been grouped because of their low execution times in comparison with the rest of the parts.

Generally, the SFM implementation always takes much less time than original LS implementation for reconstructing the same structure. In particular the first implementation has been 2.6, 5.7, 2.9, and 2.8 times slower than the SFM implementation for the studied examples and, in the most of the measurements, the SFM obtains smaller errors.

Example	**Mesh (voxels)**	**Recons. time (s)**	**Measurements**					
			A (μm)	$\frac{\textbf{error A}}{\Delta x}$	**B (μm)**	$\frac{\textbf{error B}}{\Delta x}$	**C (μm)**	$\frac{\textbf{error C}}{\Delta x}$
Convex corners	379x379x36	37.1	1593	0.59	1595	1.04	192.1	0.19
Fig. 3.7		14.2	1595	0.34	1600	0.31	193.4	0.00
Microneedles	380x380x204	181.7	38.12	1.13	8.26	0.45	314.3	0.45
Fig. 3.8		31.7	37.9	1.00	8.9	0.85	314.0	0.26
Accelerometer	501x1494x35	176.1	544.7	0.39	11.5	0.87	12.0	1.06
Fig. 3.9		61.6	544.4	0.28	11.6	0.90	12.0	1.06
Wagon wheel	542x542x66	128.2	54.80	2.67	90.25	2.99	21.96	0.22
Fig. 3.10		45.4	55.27	1.42	89.49	0.96	21.90	0.05

Table 3.3: Compilation of reconstruction examples parameters and measurements. For each example, measurements of both implementations, original LS (upper row) and SFM (lower row), are presented.

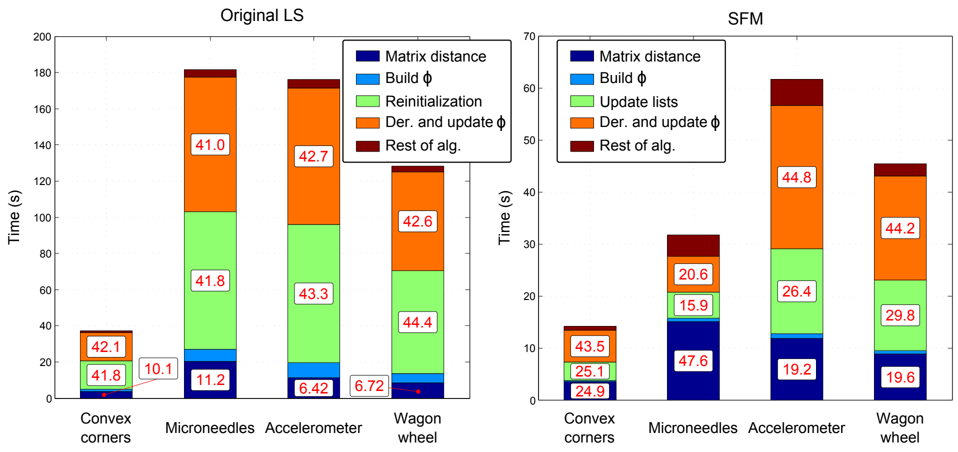


Figure 3.11: Graphs representing the contributions of the different subroutines used in both LS implementations.

3.5 Conclusions

In this chapter, two LS-based approaches have been developed to improve the visualization of wet etching CCA-based simulator results. Atomistic simulators like the CCA always provide a resulting structure formed by a cloud of unconnected atoms. This can make hard to visualize the results.

Accordingly, the application of a LS-based technique to construct a continuous surface from the cloud of points resulting from the CCA has been proposed. In particular, an energy model that depends on local curvature of surface has been used.

Two approaches have been implemented. The first is based on the original LS method, where all the mesh points are updated in every iteration and a reinitialization process is required to guarantee stability and convergence. The second implementation is a local SFM-based that updates only those points close to the surface, resulting in a more efficient approach.

First, a study has been made to determine the adequate parameters, like mesh resolution and number of iterations. Then, both implementations have been compared with the CCA results in four examples. Additionally, some features of the structures have been measured in order to characterize the error of both implementations. The errors produced by both LS implementations are usually lower than LS mesh resolution.

Despite the completely different topologies presented by the studied four examples, both LS implementations have been able to reconstruct flawlessly all of them. The final surfaces are always continuous even in those complicated parts of the structures. In general, the SFM implementation obtains more accurate results and lower errors, than the original LS-based, being capable to reconstruct even those accurate parts defined by single atoms. In addition, the SFM has been between 2.6 and 5.7 times faster than the other approach.

In conclusion, the developed LS implementations have been able to improve visual representation of CCA results applied to MEMS micromachining. Especially interesting is the SFM approach since it obtains errors usually lower than mesh resolution and the results are provided in just tens of seconds. Thus, it can be very useful to ease the design of MEMS.

Chapter 4

Level Set method for wet etching simulation

After the improvement of CCA results visualization by means of the LS method applied as a surface reconstruction process, in this chapter, the LS approach is used to simulate wet etching process. In the first place, a SFM-based algorithm that uses directly information taken from experiments is developed. Three implementations of this LS algorithm are developed. The first one is a pure sequential CPU approach, which is used for validating the proposed SFM algorithm. This approach is compared with a state-of-the-art CCA model similar in terms of computational efficiency and accuracy of results.

Furthermore, two parallel implementations of the SFM algorithm are developed, namely a multi-core CPU and a many-core GPU. A comparison between both implementations is performed, proving the excellent computational efficiency achieved when executing the SFM algorithm on a GPU.

The proposed SFM algorithm is applied to the simulation of complex MEMS micromachining processes, based on silicon and quartz substrates. These results are compared to a CCA approach, including a direct comparison with experimental results.

This chapter is structured as follows. In section 4.1 the most advanced atomistic models as well as the currently existing LS-based simulators of wet etching process are introduced. Then, section 4.2 explains how the experimental data is used by the developed simulator. Later, in section 4.3 technical details of the proposed SFM-based algorithm are given. Parallel implementations of this algorithm are introduced in section 4.4, including the CPU and the GPU versions. Finally, the results obtained with all of these implementations are shown and compared with experimental and CCA ones in section 4.5.

4.1 Introduction and drawbacks of previous wet etching simulators

Chemical wet etching is a very useful and important bulk micromachining process for MEMS fabrication as explained in section 2.4.3. The key aspects of this method are the low cost, the ability to generate smooth and flat surfaces, and to release suspended structures. Nevertheless, the shape of the resulting structures is difficult to predict due to multiple experimental parameters, such as crystallographic orientations [176, 191, 276, 277], composition of etchant solution (e.g. KOH [188, 193] or TMAH [177, 278]), its concentration and temperature [191, 192], and usage of additives (e.g. Triton [175, 195] or IPA [193, 194]). As a consequence, an important effort has been made in last years to model the process accurately in order to avoid the need to perform several experiments until the desired structure is achieved.

4.1.1 CCA wet etching simulators

Although several techniques have been used to simulate wet etching process (see section 2.4.3.3), atomistic models and, in particular the CCA, are currently considered the most accurate and reliable [102, 180, 187].

This technique emulates the etching process by, first, constructing a mesh of atoms according to the crystallographic structure of the substrate. Then, the surface atoms that are in contact with the etchant solution are continuously removed depending on their neighbourhood configuration (see section 2.4.3.3). According to this, Gosálvez et al. introduced the *step-flow* model [219]. The step-flow proposes that the high index silicon planes (e.g. (533)) are formed by several (111) terraces separated by steps. These terraces start to retract since the atoms at the step are weakly connected to bulk atoms and they are much easier to be removed [218]. This study provided an extensive classification of the atom configurations existing between the tree main orientations $(001), (110)$, and (111), i.e. each of these atoms was classified regarding the number of first and second surface and bulk neighbouring atoms, namely $n^{1s}, n^{2s}, n^{1b}, n^{2b}$. In the CCA, every atom has an internal value of occupancy Π, which initially is 1 and after one iteration it is reduced such that, when occupancy is equal or lower than 0, an atom is removed. Thus, according to the current configuration of each atom, the corresponding amount of occupancy reduction is applied, i.e. for every atom, the occupancy at time step k is updated with:

$$\Pi^{k+1} = \Pi^k - \Delta t \cdot r^k, \tag{4.1}$$

where Δt is the time step and:

$$r^k = R(n^{1s}, n^{2s}, n^{1b}, n^{2b}), \tag{4.2}$$

is the amount of occupancy that has to be removed, which is determined by the function R that relates atomistic configuration with etch rate. Function R depends on etchant solution (e.g. KOH or TMAH)and its temperature and concentration.

Nevertheless, the issue is that etch rates are experimentally obtained in a macroscopic form, i.e. a specific orientation plane is etched at a specific etch rate. Thus, the other main contribution of Gosálvez et al. was to develop a system of equations relating macroscopic etch rates with atomistic configurations [219].

However, since n^{1s} and n^{1b} can take values between 0 to 3, and n^{2s} and n^{2b} between 0 to 11 [187, 217, 219], there are 4x4x12x12 = 2304 possible R values, which is higher than independent equations. Thus, initially, a manual calibration was required depending on etchant and experimental conditions to determine only 33 removal rates R values, which is sufficient to simulate different etchants like KOH, KOH+IPA and TMAH [219, 220, 279].

Nevertheless, there are still atomistic configurations that are not classified by the step-flow model. In order to address this problem, in the first place, a removal probability function that directly provides a removal rate regarding first and second number of neighbouring atoms was used [220, 280]. However, this method was not able to reproduce the usage of additives and requires a manual calibration process, which can become a complicated task, especially for etchants with a significantly different behaviour than KOH [222].

Accordingly, the usage of Evolutionary Algorithm (EA) in order to find the values for the rest of atomistic configurations that were not included in the step-flow classification was proposed [222]. An EA is an optimization trial-and-error process that starts from an initial set of possible solutions. Every solution is formed by a set of parameters which are changed after each iteration. In CCA models, these parameters are the amount of occupancy reduction for each atomistic configuration. Thus, every possible solution is evaluated according to several objective functions, including differences between experimental and simulated etching of a sphere, discrepancies with experimental measurements when etching a (100) surface protected with a circular mask, and differences between etch rate of similar orientations (due to crystallographic symmetries). Those possible solutions that obtain better errors, have more probabilities to be chosen to be combined with other solutions and, thus, producing new solutions. This process is repeated and tends to obtain solutions with less error after each iteration. When the errors are small enough, the best solution is selected.

Accordingly, the calibration CCA process of a specific etchant (solution, concentration and temperature) by means of an EA is formed by the following steps:

1. Generate all the possible solutions.
2. Run a CCA simulation for every possible solution.
3. Calculate the errors of the different objective functions.
4. Generate the new possible solutions according to the errors.
5. Go to step 1 until convergence is achieved.

Usually, a calibration process requires several hundreds or even thousands of iterations to successfully calibrate the CCA. This laborious task can take advantage of distributed and parallel environments but, even so, it can take between a few hours to several tens of hours [187, 222]. However, thanks to accurate CCA models and the robustness of EA, more than 30 silicon etchants have been calibrated and can be used to perform CCA wet etching simulations [222]. Additionally, EAs in combination with CCA have been used to calibrate and simulate NH_4HF_2 for wet etching of quartz substrates [180].

Despite the step-flow CCA-based approach in combination with the EA calibration process has been able to simulate silicon and quartz wet etching in many etchant solutions, the main drawbacks are:

- The resulting structure is formed by a cloud of points, making necessary to use additional techniques (such as the presented in chapter 3) to visualize correctly the results.
- It is necessary the knowledge of the substrate structure in order to make a thorough classification of the different atomistic configurations.
- A calibration process is required to relate experimental macroscopic etch rates with atomistic removal rates. This process is required each time the experimental conditions are changed, i.e. for each etchant solution, concentration, temperature, usage of additives, or substrate material.

In this context, the search for an alternative method, which is simultaneously capable of using experimental data without any prior calibration while remaining computationally efficient, has the potential to significantly influence the future of MEMS design. Thus, in this thesis, the LS method has been considered as an alternative to the CCA approach.

4.1.2 Wet etching simulation by means of the LS method

Although atomistic approaches have been widely used to simulate chemical wet etching, the LS method has also been applied previously to emulate this process. In particular, Adalsteinsson and Sethian introduced a unified model for etching, deposition and lithography in 1995 [16–18]. Despite only simple etching processes were considered (e.g. isotropic and unidirectional etching processes), they proved the capability of the LS for the simulation of such micromachining processes [281].

Later, Radjenović et al. presented a LS implementation to simulate silicon wet etching with KOH 30 wt% at 70 °C [223, 282]. This simple approach only considered the experimental etch rates for the main orientations (111), (110), and (100) whereas, for the rest of the orientations, an interpolation relation was used. Nevertheless, the SFM was validated for wet etching simulation. Finally, they developed a new implementation that allows the simulation of silicon wet etching with KOH with concentrations 30 wt%, 40 wt%, and 50 wt% at 70 °C. For every etchant solution, 13 experimental etch rate values were considered [225]. This approach was capable to simulate silicon etching with several simple masks, demonstrating the possibility of evolving anisotropic etching fronts with the LS method. On the other hand, this results have been never compared with other existing methods, such as the CCA approach, nor with experimental results. Additionally there is still much room for further optimizations and improvements.

Accordingly, a more exhaustive evaluation is required, therefore, in contrast with the already developed LS wet etching simulators, in this thesis several SFM-based implementations are developed, including a sequential algorithm as well as a CPU and a GPU parallel implementations. Moreover, full distributions of 8100 experimental etch rates are used. The reported simulations include many etchant solutions, such as KOH, KOH+IPA, TMAH, TMAH+Triton, and NH_4HF_2 for quartz substrates, at different concentrations and temperature. Furthermore, the possibility to perform complex experiments, such as simulating etching processes on double-sided wafers of any material as well as advanced processes, including the sequential use of different mask patterns during successive etching steps with the possibility of changing top and/or bottom masks as well as etchant solution.

Finally, these algorithms are compared to each other in terms of computational efficiency as well as an exhaustive comparison with the state-of-the-art CCA approach in terms of accuracy and computational efficiency is performed. Additionally, the simulation results are directly compared with those from experiments.

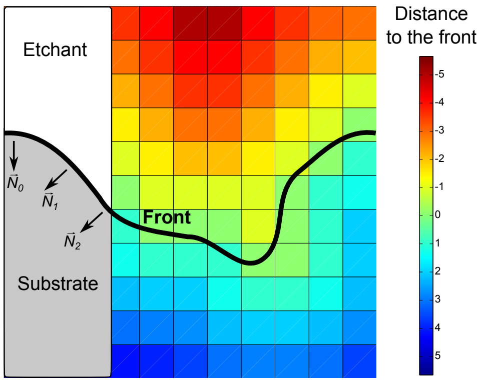


Figure 4.1: Two-dimensional example of an etching process simulation by means of the LS method. The front (black line) embedded inside a SDF corresponds to the substrate surface in contact with the etchant solution. Several local normal vectors, which are utilized to determine local etch rates, are represented. Reproduced from [19].

4.2 Experimental etch rates

The LS method is a numerical technique to track moving fronts according to some physical motions which depend on the phenomenon being emulated. When emulating chemical wet etching process, the moving front corresponds with the surface of the material that is in contact with the etchant. In addition, the motion of the front is determined by many external factors, such as etchant, its temperature, concentration and possible additives, and substrate material. Moreover, anisotropic wet etching is a relevant process in the field of MEMS micromachining because of the strong dependency of etch rates on crystallographic structure of substrate material. Hence, the velocity of the front is determined by external factors but, also, it depends on the topography of the front itself. According to LS technique, the substrate surface in contact with etchant solution is the three-dimensional surface embedded inside a SDF and evolved according to experimental etch rates. A simple two-dimensional example is depicted in Fig. 4.1. Notice that, contrary to LS implementation of chapter 3, the LS front is not a closed front (or closed surface in three-dimensions). Furthermore, in this chapter, the distance values of those grid points lower to the front are considered positive whereas the upper points are signed as negative, nevertheless, this has no effect on the LS behaviour.

The developed implementations, utilize directly experimental etch rates to move the surface being etched. These etch rates values are usually obtained by etching and measuring structures that resemble a wagon wheel [190, 191]. These type of experiments produces many spokes, each of them has been etched a

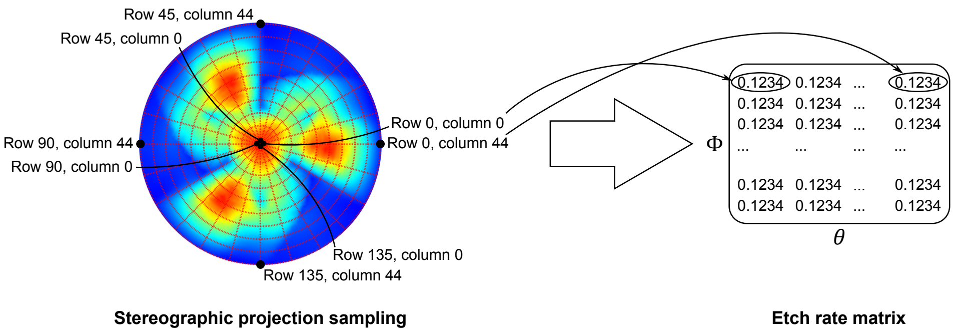


Figure 4.2: Example of an etch rate distribution discretization and arranging process in matrix form. This matrix can be accessed with spherical coordinates (θ, Φ).

specific distance (see Fig. 2.28) depending on the corresponding crystallographic orientation. By measuring all the produced spokes, a specific etchant and conditions are characterized obtaining a complete etch rate distribution.

From an experimental point of view, the traditional way to represent the anisotropy of an etchant is to show a stereographic projection of the unit sphere while presenting the etch rates at different locations (θ, Φ) using isolines and/or a collection of colors (see section 2.4.3.2). Since a sphere contains all possible substrate orientations, it provides a complete representation of the etchant anisotropy in a compact way. The procedure originated in early experimental reports, where real hemispherical samples were actually etched [196, 196]. By mechanically probing the hemisphere surface before and after etching at different latitude/longitude locations, the etched distance is determined as a function of the orientation, in terms of latitude (θ) and azimuth (Φ) spherical coordinates, thus obtaining the etch rate corresponding to a normal vector by dividing by the etch time.

Accordingly, in this thesis it has been possible to use an extensive data base of experimental etch rate distributions of 33 different etchants obtained by Gosálvez et al. [191]. These distributions are hemispheres discretized with an angular resolution of 2° for both spherical coordinates θ, Φ, resulting in 8100 etch rate values (etched micrometers per minute) arranged in a 180x45 matrix $R(\theta, \Phi)$. This discretizing and arranging process is depicted in Fig. 4.2, showing the relation between stereographic projection and etch rate matrix. Since only hemispheres are considered, the angular domain of both coordinates is:

$$\begin{aligned} \theta &= [0,\ 90] \\ \Phi &= [0,\ 358]. \end{aligned} \tag{4.3}$$

These spherical coordinates depend on surface local geometry, in particular, they can be obtained from local normal vector $\vec{N} = (N_x, N_y, N_z)$:

$$\theta = \begin{cases} \arctan\left(\frac{\sqrt{N_x^2+N_y^2}}{N_z}\right) & \text{if } N_z > 0 \\ \arctan\left(\frac{\sqrt{N_x^2+N_y^2}}{N_z}\right) + \pi & \text{if } N_z < 0 \\ \frac{\pi}{2} & \text{otherwise} \end{cases}$$

$$\Phi = \begin{cases} 2\pi - \arctan\left(\frac{N_y}{N_x}\right) & \text{if } N_x > 0, N_y > 0 \\ \arctan\left(\frac{-N_y}{N_x}\right) & \text{if } N_x > 0, N_y \leq 0 \\ \pi - \arctan\left(\frac{N_y}{N_x}\right) & \text{if } N_x < 0 \\ \frac{\pi}{2} & \text{if } N_x = 0, N_y < 0 \\ \frac{3\pi}{2} & \text{otherwise} \end{cases} \tag{4.4}$$

Since only a hemisphere is considered, if $\theta > \frac{\pi}{2} \rightarrow \theta = \pi - \theta$. Notice normal vector of an implicit function ϕ can be obtained with (2.25). Accordingly, the three-dimensional components are determined with:

$$\begin{aligned} N_x &= \frac{\phi_x}{|\nabla\phi|} \\ N_y &= \frac{\phi_y}{|\nabla\phi|} \\ N_z &= \frac{\phi_z}{|\nabla\phi|}. \end{aligned} \tag{4.5}$$

By using (4.4) and the definition of the normal vector (2.25) for a SDF, surface local etch rates can be obtained from the LS function ϕ. Hence, the etch rate matrix can be understood as $R(\vec{N}) = R(N_x, N_y, N_z)$, which is the equivalent to the velocity field $\vec{V}$ of the LS method introduced in section 2.2. Therefore, each element of the surface being etched will be advanced according to its corresponding etch rate value $R(\vec{N})$. Because of the motion is in the normal vector direction, the LS equation results in:

$$\phi_t + R(\vec{N})|\nabla\phi| = 0. \tag{4.6}$$

Thus, it is easy to realize that the etch rates that must be applied to advance the surface, depend directly on local topology of surface, hence, numerical schemes described in section 2.2.2 must be utilized to solve (4.6). Furthermore, Hamiltonian of (4.6) is defined as:

$$H(\phi) = R(\vec{N})|\nabla\phi|. \tag{4.7}$$

Because of the dependency of the Hamiltonian on the normal vector $\vec{N}$, this is a non-convex Hamiltonian since it does not fulfil condition (2.47) [16, 281]. As a consequence, upwind differencing technique cannot be utilized (see section 2.2.2). In its place, in this thesis the LFS is used to solve (4.6) since, based on central difference derivatives, LFS is the simplest scheme that preserves monotonicity. This scheme replaces the Hamiltonian from (4.6) with a new Hamiltonian $\hat{H(\phi)}$ defined in (2.48). Particularly in this study, the next expression has been employed:

$$\begin{aligned} \hat{H(\phi)} = \quad & H\left(\tfrac{\phi_x^+ + \phi_x^-}{2}, \tfrac{\phi_y^+ + \phi_y^-}{2}, \tfrac{\phi_z^+ + \phi_z^-}{2}\right) \\ & -\alpha_F \left(\alpha_x \tfrac{\phi_x^+ - \phi_x^-}{2} + \alpha_y \tfrac{\phi_y^+ - \phi_y^-}{2} + \alpha_z \tfrac{\phi_z^+ - \phi_z^-}{2}\right) \end{aligned}, \tag{4.8}$$

resulting in the LS equation:

$$\phi_t + R(\vec{N})|\nabla\phi| - \alpha_F \left(\alpha_x \frac{\phi_x^+ - \phi_x^-}{2} + \alpha_y \frac{\phi_y^+ - \phi_y^-}{2} + \alpha_z \frac{\phi_z^+ - \phi_z^-}{2}\right) = 0, \tag{4.9}$$

where $\alpha_F \geq 0$ is a numerical value introduced to control the overall viscosity and guarantee the stability of the front by modifying simultaneously the three artificial viscosity factors α_x, α_y and α_z. A larger α_F results in smoother surfaces while smaller α_F generates sharp features/discontinuities, which should be handled carefully to avoid unrealistic and/or diverging solutions. Depending on the anisotropy of the simulated etchant, different values of α_F are needed. In this study it has been found that $\alpha_F = 0.48$ is adequate to simulate KOH-based etchants, while $\alpha_F = 0.45$ and $\alpha_F = 0.5$ are more suitable to simulate TMAH-based and saturated NH_4HF_2-based etchants, respectively. The procedure to obtain these parameters is trial and error, selecting those values which produce the most realistic results. Too large values result in soft, featureless fronts, while too small values result in discontinuities and/or unnatural, sharp features.

The artificial viscosity factors are defined in their general form by (2.49). For simplicity, only expressions of spatial dimension x are derived step by step, nevertheless, for the rest of dimensions the process is analogous. According to (2.49), the viscosity factor can be written as:

$$\alpha_x = \max \left| \frac{\partial R(\vec{N})}{\partial \phi_x} \cdot |\nabla\phi| + \frac{\partial |\nabla\phi|}{\partial \phi_x} \cdot R(\vec{N}) \right|. \tag{4.10}$$

Due to $R(\vec{N})$ depends directly on normal vector instead of spatial derivatives of ϕ, the chain rule must be applied to the first term, leading to:

$$\frac{\partial R(\vec{N})}{\partial \phi_x} = \frac{\partial R(\vec{N})}{\partial N_x} \cdot \frac{\partial N_x}{\partial \phi_x}. \tag{4.11}$$

In turn, considering:

$$\frac{\partial |\nabla\phi|}{\partial \phi_x} = \frac{\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}}{\partial \phi_x} = \frac{1}{2}(\phi_x^2 + \phi_y^2 + \phi_z^2)^{-1/2} \cdot 2\phi_x = \frac{\phi_x}{|\nabla\phi|} = N_x, \tag{4.12}$$

and applying the quotient rule:

$$\begin{aligned}\frac{\partial N_x}{\partial \phi_x} &= \frac{\phi_x/|\nabla\phi_x|}{\partial\phi_x} = \frac{|\nabla\phi| - \phi_x\frac{\partial|\nabla\phi|}{\partial\phi_x}}{|\nabla\phi|^2}\\ &= \frac{|\nabla\phi| - \phi_x(\phi_x/|\nabla\phi|)}{|\nabla\phi|^2} = \frac{1}{|\nabla\phi|} - \frac{\phi_x^2}{|\nabla\phi|^3}\\ &= \frac{\phi_y^2+\phi_z^2}{|\nabla\phi|^3}\end{aligned} \tag{4.13}$$

Thus, by taking (4.13) into (4.11):

$$\frac{\partial R(\vec{N})}{\partial\phi_x} = \frac{\partial R(\vec{N})}{\partial N_x}\frac{\phi_y^2+\phi_z^2}{|\nabla\phi^3|}. \tag{4.14}$$

Finally, by replacing the expressions of (4.14) and (4.12) into (4.10) and proceeding analogously for the rest of dimensions, the viscosity factors are obtained:

$$\begin{aligned}\alpha_x &= \max\left|\frac{\partial R(\vec{N})}{\partial N_x}\frac{\phi_y^2+\phi_z^2}{|\nabla\phi^3|}\cdot|\nabla\phi| + N_x\cdot R(\vec{N})\right|\\ \alpha_y &= \max\left|\frac{\partial R(\vec{N})}{\partial N_y}\frac{\phi_x^2+\phi_z^2}{|\nabla\phi^3|}\cdot|\nabla\phi| + N_y\cdot R(\vec{N})\right|\\ \alpha_z &= \max\left|\frac{\partial R(\vec{N})}{\partial N_z}\frac{\phi_y^2+\phi_x^2}{|\nabla\phi^3|}\cdot|\nabla\phi| + N_z\cdot R(\vec{N})\right|\end{aligned} \tag{4.15}$$

As can be observed, these factors depend on the variation of etch rate with respect to normal vector components, i.e. the terms $\frac{R(\vec{N})}{\partial N_q}$ for $q = x, y, z$ dimensions. In order to numerically obtain these derivatives, it is necessary to discretize the components of the normal vector. In this thesis it has been found experimentally that good results are obtained by using second-order central differences with a resolution of 0.01. Thus, these terms are calculated with the next equations:

$$\begin{aligned}\frac{R(\vec{N})}{\partial N_x} &= \frac{R(N_x+0.01, N_y, N_z) - R(N_x-0.01, N_y, N_z)}{2\cdot 0.01}\\ \frac{R(\vec{N})}{\partial N_y} &= \frac{R(N_x, N_y+0.01, N_z) - R(N_x, N_y-0.01, N_z)}{2\cdot 0.01}\\ \frac{R(\vec{N})}{\partial N_z} &= \frac{R(N_x, N_y, N_z+0.01) - R(N_x, N_y, N_z-0.01)}{2\cdot 0.01}\end{aligned} \tag{4.16}$$

Notice that despite etch rate matrix R has been written as $R(\vec{N})$ to emphasize the dependency on normal vector, the actual etch rate matrix is defined for spherical coordinates, i.e. $R(\theta, \Phi)$. Moreover, because of etch rates are only defined for discrete spherical coordinates (a total of 8100 values) and normal vectors are not restricted to specific values, an approximation technique must be applied to select the most suitable $R(\vec{N})$ corresponding values. However, this approximation may introduce abrupt changes, causing numerical instabilities. In order to prevent this, an interpolation process is employed. Consider a specific normal vector $\vec{N}_0 = (N_{x0}, N_{y0}, N_{z0})$. For this vector, the corresponding latitude (θ_0) and azimuth (Φ_0) values are calculated with (4.4) and then they are normalized according to the domain (4.3). Then, the next weights are determined:

$$\begin{aligned} P_1 &= R(\lfloor\theta_0\rfloor, \lfloor\Phi_0\rfloor) \\ P_2 &= R(\lfloor\theta_0\rfloor, \lceil\Phi_0\rceil) \\ P_3 &= R(\lceil\theta_0\rceil, \lfloor\Phi_0\rfloor) \\ P_4 &= R(\lceil\theta_0\rceil, \lceil\Phi_0\rceil) \end{aligned}, \tag{4.17}$$

and the next distance values are calculated:

$$\begin{aligned} D_1 &= \theta_0 - \lfloor\theta_0\rfloor \\ D_2 &= \Phi_0 - \lfloor\Phi_0\rfloor \end{aligned}. \tag{4.18}$$

These values are depicted in Fig. 4.3. Finally, the etch rate value corresponding to $\vec{N}_0$ is calculated as:

$$\begin{aligned} R(\vec{N}_0) = \quad & P_1((1 - D_1) \cdot (1 - D_2)) \\ & +P_2((1 - D_1) \cdot (D_2)) \\ & +P_3((D_1) \cdot (1 - D_2)) \\ & +P_4((D_1) \cdot (D_2)). \end{aligned} \tag{4.19}$$

Notice that, thanks to the LS method and the direct usage of experimental etch rate distributions, etching conditions such as: temperature, surface orientation, substrate material (in this thesis, both quartz and silicon substrates are modelled), etchant concentration, and the inclusion of additives (e.g. IPA or Triton) are modelled exclusively by using the corresponding etch rate matrix $R(\theta, \Phi)$.

4.2.1 Isotropic etchant

Isotropic etching is used as a complementary process in connection with anisotropic etching and/or DRIE [283]. For isotropic etchants, the etch rate remains essentially constant along any direction. When an isotropic etching process is simulated, some simplifications can be made. An isotropic etching can be understood as an anisotropic etching with $R(\theta, \Phi) = 1$ for every possible θ and Φ angles. This case corresponds to a front moving in its normal direction at constant velocity. Such cases cannot be properly solved with upwind differencing since backward

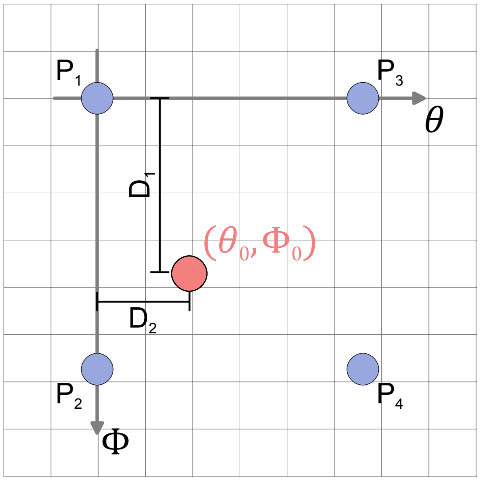


Figure 4.3: Interpolation applied to approximate the proper etch rate value for a generic (θ_0, Φ_0) values, according to the known etch rate values P_1, P_2, P_3 and P_4.

and forward spatial derivatives $(\phi_q^{\pm})$ can have different signs at sharp features of the front [33]. This issue is solved by using LFS as well.

The main feature of isotropic etching simulation by means of LFS is that it is not necessary to search the values for the artificial viscosity coefficients of (4.9) because of etch rate distribution R is always constant and, thus, the derivatives $\frac{\partial R(\vec{N})}{\partial N_q} = 0$, implying $\alpha_q = 1$ in all dimensions. Consequently, the next equation is applied when an isotropic etchant is simulated:

$$\phi_t + |\nabla\phi| - \left(\frac{\phi_x^+ - \phi_x^-}{2} + \frac{\phi_y^+ - \phi_y^-}{2} + \frac{\phi_z^+ - \phi_z^-}{2}\right) = 0. \tag{4.20}$$

4.3 SFM developed simulator

Last section presents how the LS method is adapted in this thesis to simulate wet etching process using directly a complete etch rate distribution. In this section, the steps of the developed algorithm are detailed. In this chapter, only SFM-based implementations are considered since the SFM presents more accurate results than original LS method (which computes the whole grid space) as well as faster simulations due to computational cost is reduced to roughly $O(N^2)$ by only computing those grid points close to the surface.

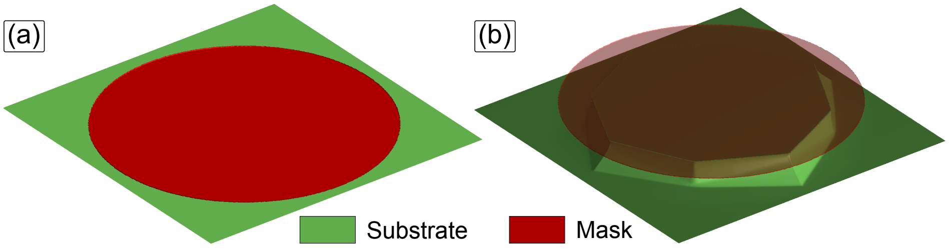


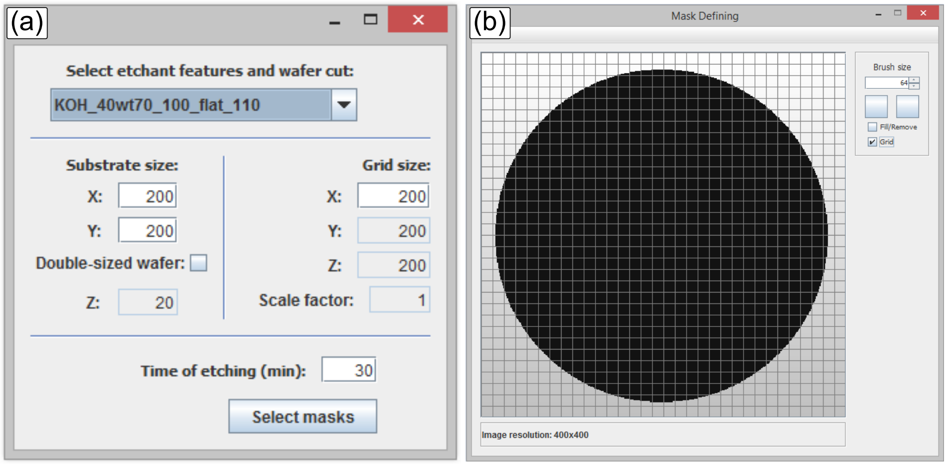
Figure 4.4: Simple etching process simulation: (a) initial substrate flat surface and the applied circular mask, (b) resulting structure after etching process.

In the first place, the simulation of simple etching process is explained. A simple etching process consists of a flat surface etched for a concrete time with a single mask that remains constant for the whole etching process. An example of simple etching process is shown in Fig. 4.4. Notice that in all the performed simulations, the applied masks are considered ideal, i.e. masks are never etched.

In the developed LS-based simulation tool, user needs to introduce the next input parameters to perform an etching process simulation:

- Surface substrate size in micrometers: $subs_x$ and $subs_y$ for the corresponding dimensions x and y.
- LS mesh dimension.
- Time of the etching process T_e.
- Etchant solution: concentration, temperature and additives.
- Surface wafer orientation.
- Mask to deposit on wafer surface. This simulates the lithographic process used to mask the wafer.

Fig. 4.5 shows the input panels of the simulation tool that allow user to define the etching process properties. Notice that etch rate distributions used by the LS method depend not only on the etchant features but also on the wafer surface orientation since this crystallographic surface orientation must correspond to the orientation at the center of the stereographic projection, i.e. to $R(\theta = 0, \Phi = 0)$. Nevertheless, translations and rotations can be performed to transform a specific etch rate matrix to another with a different surface orientation.


(a)

Select etchant features and wafer cut: KOH_40wt70_100_flat_110

Substrate size:
X: 200
Y: 200
Double-sized wafer: ☐
Z: 20

Grid size:
X: 200
Y: 200
Z: 200
Scale factor: 1

Time of etching (min): 30

(b)

Mask Defining

Brush size 64
☐ Fill/Remove
☑ Grid

Image resolution: 400x400

Figure 4.5: Input parameters panels: (a) substrate and grid size definitions, time of etching, and wafer and etchant features, (b) mask definition panel that allows to load a predefined mask.

4.3.1 Mesh generation

LS mesh represents the substrate and its surface in contact with the etchant, as shown in Fig. 4.1. A regular grid is used in this thesis, thus grid resolution is the same for all dimensions $\Delta x = \Delta y = \Delta z$. Notice that grid resolution affects directly the accuracy of simulation results (as commented in section 4.5.1). The surface grid size is determined according to input parameters introduced by user. Accordingly, the number of grid points in x dimension (num_x) is directly introduced by the user whereas the number of points in y is determined by:

$$num_y = \frac{num_x}{subs_x} subs_y. \tag{4.21}$$

However, the number of points in z dimension (num_z) must be determined to ensure that is sufficiently large to perform the whole etching process, otherwise the moving front would disappear at those places that reach the bottom of the grid. Furthermore, if num_z is too large, an unnecessary amount of memory would be utilized. Notice that, since the SFM is utilized, computational time depends mainly on the number of points that form the surface, rather than on grid size. However, the whole grid is allocated in memory space and it should be as small as possible. Accordingly, num_z is calculated regarding the etch rate in the vertical direction and the time of etching:

$$num_z = 4 + T_e \cdot R(\theta = 0, \Phi = 0). \tag{4.22}$$

4 additional points are introduced to properly handle the SFM lists. By calculating num_z with this equation, an adequate amount of memory is used. Notice that, if an isotropic etching process is simulated, $R(\theta = 0, \Phi = 0) = 1\mu\text{m/min}$ is assumed.

4.3.2 Initial surface determination

Once the LS grid has been generated according to etching and user input parameters, initial surface must be determined. For simple etching processes, initial surface is simply a horizontal plane on the top of the grid. Particularly, if the number of grid points in vertical dimension is num_z and indices start at 0, the initial surface is the plane formed by the points $(i, j, k = num_z - 2)$.

According to the LS method, this surface must be embedded inside a SDF, however, due to SFM, only those adjacent points to the surface must be updated and included in the SFM lists depending on their distance values (see table 2.1). According to the LFS applied to the LS equations (4.9) and (4.20), second-order central differences must be utilized to calculate spatial derivatives of SDF ϕ in order to calculate normal vector components (4.5) as well as gradient $|\nabla\phi|$. Hence, only three SFM lists are required to update active grid points, namely L_0, L_{+1} and L_{-1}. Additionally, each grid point is tagged with $state = 0, 1, -1, 2, -2$ depending on the distance value as shown in table 2.1.

Moreover, when simulating wet etching processes those surface points covered with masking material deposited on the top of the substrate must be kept to constant values in order to emulate the physical process. The mask is loaded from an image file and it is rescaled according to surface grid dimensions, producing a two-dimensional matrix $mask_t$ with $num_x \cdot num_y$ size, such that $mask_t(i, j) = 1$ indicates that the surface grid point (i, k) is covered by the mask and $mask_t(i, j) = 0$ is used for non-masked grid points. Accordingly, algorithm 14 is used to generate initial surface, its SDF, the SFM lists and the *state* tag for every grid point. Correspondingly, Fig. 4.6 depicts a two-dimensional example showing the initial state of the grid points according to a simple mask.

4.3.3 Numerical discretization

One of the input parameters is the etching time T_e. This total etching time must be discretized into time steps in order to guarantee stability when updating (4.9) or (4.20). In particular, a forward first-order Euler time discretization is employed, thus, ϕ values at time $t = (n+1)\Delta t$ are obtained with:

$$\begin{aligned}\phi^{n+1} \quad &= \phi^n - \Delta t \Big[R(\theta, \Phi)|\nabla\phi^n| \\ &-\alpha_F \left(\alpha_x^n \tfrac{(\phi_x^+)^n - (\phi_x^-)^n}{2} + \alpha_y^n \tfrac{(\phi_y^+)^n - (\phi_y^-)^n}{2} + \alpha_z^n \tfrac{(\phi_z^+)^n - (\phi_z^-)^n}{2} \right) \Big],\end{aligned} \tag{4.23}$$

Algorithm 14: Simple initial top surface determination.

```
1  Initialize every grid point to $\phi = -1.5\Delta x$ and $state = -2$.
   for each grid point $(i, j, k = num_z - 2)$ do
2      $\phi(i, j, k = num_z - 2) = 0$.
3      $state(i, j, k = num_z - 2) = 0$.
       if $mask_t(i, j) = 0$ then
           add $(i, j, k = num_z - 2)$ to $L_0$.
           add $(i, j, k = num_z - 1)$ to $L_{-1}$.
           add $(i, j, k = num_z - 3)$ to $L_{+1}$.
4          $\phi(i, j, k = num_z - 1) = -\Delta x$.
5          $state(i, j, k = num_z - 1) = -1$.
6          $\phi(i, j, k = num_z - 3) = \Delta x$.
7          $state(i, j, k = num_z - 3) = 1$.
       else
8          $\phi(i, j, k = num_z - 1) = -1.5\Delta x$.
9          $state(i, j, k = num_z - 1) = -2$.
10         $\phi(i, j, k = num_z - 3) = 1.5\Delta x$.
11         $state(i, j, k = num_z - 3) = 2$.
```

where Δt is the time step, which must be obtained with the corresponding CFL condition. Since this LS equation does not depend on second derivatives (such as local curvature), a time step $\Delta t = O(\Delta x)$ according to (2.45) can be employed. Taking into account that $\max\{|\nabla\phi|\} = 1$ by definition, it has been found after numerous simulations that:

$$\Delta t = \frac{0.3\Delta x}{\max\{R(\theta, \Phi)\}} \tag{4.24}$$

is a proper value that ensures realistic results for every tested etch rate distribution. Since etch rates are provided in $[\mu\text{m}]/[\text{min}]$, Δt is also in minutes. Hence, the number of iterations needed to perform an etching process of the introduced etching time (also in minutes) can be directly calculated by:

$$N_{iter} = \frac{T_e}{\Delta t}. \tag{4.25}$$

Likewise, due to LFS (4.8) utilized in order to evolve the surface properly, the terms $\nabla\phi$ and $\vec{N}$ must be calculated with second-order central differences (2.28), whereas for forward and backward spatial derivatives $\phi_q^{\pm}$ simple first-order differences are utilized ((2.26) and (2.27) respectively).

Analogously, when simulating an isotropic etchant, expressions (4.23) and (4.24) can be simplified by considering $R(\theta, \Phi) = 1$ and $\alpha_q = 1$.

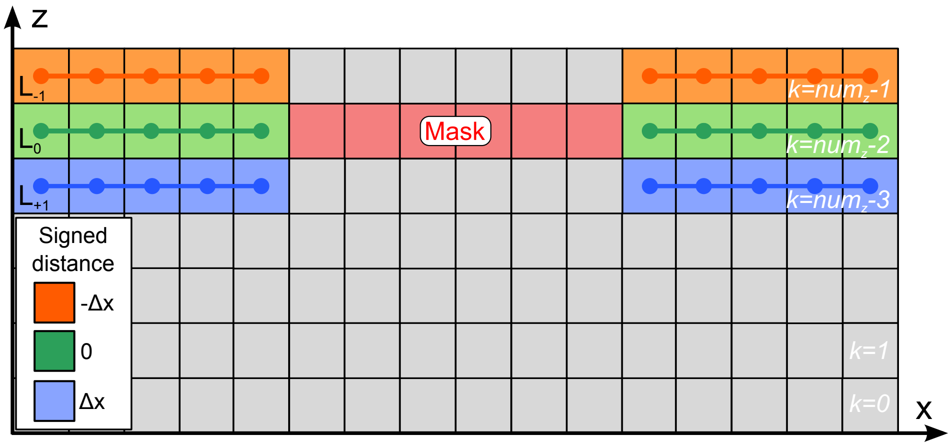


Figure 4.6: Two dimensional example of an initial flat surface partially covered with a mask. Only grid points that are not covered with the mask are included in the corresponding SFM lists. Orange, green and blue voxels represent the L_{-1}, L_0 and L_{+1} lists used in the SFM, respectively, while gray and red voxels are not included in any list since they do not need to be updated. Positive/negative distances are associated to the points located below/above the front. Signed distance values are shown.

4.3.4 Complete algorithm

When input parameters has been introduced, the mask pattern has been loaded, and the mesh and initial surface have been generated accordingly, the etching process is simulated by updating the ϕ values of grid points with (4.23) for the corresponding number of iterations depending on the desired time of etching. The workflow of the developed SFM simulator can be observed in algorithm 15.

It is important to notice that viscosity factors α_q must be calculated in each iteration, which implies the search of the maximum values (4.15). As a consequence, the first L_0 loop (step 7) must be completely performed before updating ϕ values (second L_0 loop, step 13). However, this search is limited to only those surface points included in L_0 list.

The main variables of the algorithm are collected in table 4.1. Notice that, despite most of the variable are stored for every mesh point, only those values included in SFM lists are computed. On the other hand, the three-dimensional variable $mask3D$ will be useful when simulating more complex processes since not only points of initial surface can be masked, thus a value indicating whether a grid point is masked or not will be useful. By knowing the amount of memory required by all the main variables, theoretical amount of memory taken by the algorithm can be calculated, which is useful for comparing to another existing methods.

Algorithm 15: SFM-based algorithm for the simulation of simple wet etching process.

```
 1 Introduce etching process features: substrate type (silicon or quartz), crystallographic
   orientation, etchant concentration, temperature, etching time, substrate size, and grid
   size.
 2 Load the mask pattern.
 3 Generate the LS mesh according to input parameters. Eq. (4.22) is used to determine
   the vertical size of the mesh.
 4 Build initial surface, its SDF, and the SFM lists according to algorithm 14.
 5 Obtain $\Delta t$ according to (4.24) and the corresponding number of iterations with (4.25).
                                                                        -Etching loop-
 6 for iter=0; iter< $N_{iter}$; i++ do
 7     for every point in $L_0$ do
 8         Calculate forward and backward spatial derivatives $\phi_q^{\pm}$ for $q = x, y, z$ dimensions.
 9         Calculate second-order central differences $\phi_q = \frac{\phi_q^+ + \phi_q^-}{2}$.
           if Anisotropic etchant then
10             Determine the normal components (4.5).
11             Convert the normal vector to spherical coordinates $(\theta, \Phi)$ with (4.4) and
               obtain the corresponding etch rate value by accessing the matrix $R(\theta, \Phi)$.
12             Determine $\alpha_q$ coefficients with (4.15) and select the maximum values.
13     for each $L_0$ point $\vec{x}_i$ do
14         Update $\phi(\vec{x}_i)$ using (4.23).
           if $\phi(\vec{x}_i) < -0.5\Delta x$ then
15             remove $(\vec{x}_i)$ from $L_0$ and add it to $S_{-1}$.
           if $\phi(\vec{x}_i) > 0.5\Delta x$ then
16             remove $(\vec{x}_i)$ from $L_0$ and add it to $S_{+1}$.
17     Update $L_{+1}$ and $L_{-1}$ lists with procedure 5.
18     Transfer points from auxiliary lists by applying procedure 6.
19     Add corresponding points to $L_{\pm 1}$ according to procedure 7.
20 Extract implicit surface for visualization.
```

Variable	Type	Size	Description
ϕ	float	grid	Signed distance values of every mesh point.
ϕ_x^-	float	grid	First-order backward spatial derivative of ϕ in x.
ϕ_x^+	float	grid	First-order forward spatial derivative of ϕ in x.
ϕ_y^-	float	grid	First-order backward spatial derivative of ϕ in y.
ϕ_y^+	float	grid	First-order forward spatial derivative of ϕ in y.
ϕ_z^-	float	grid	First-order backward spatial derivative of ϕ in z.
ϕ_z^+	float	grid	First-order forward spatial derivative of ϕ in z.
H	float	grid	Hamiltonian value of every mesh point.
$mask3D$	boolean	grid	Indicates those masked grid points.
$state$	char	grid	Indicates the SFM state of every grid point.
R	float	180x45	Etch rate distribution.
$mask_t$	boolean	surface	Top mask pattern introduced by user.
$mask_b$	boolean	surface	Bottom mask pattern introduced by user.

Table 4.1: Main variables utilized in wet etching SFM implementation. The variables with size *grid* are three-dimensional matrices of $num_x \cdot num_y \cdot num_z$ points. On the other hand, *surface* size is used for two-dimensional matrices of $num_x \cdot num_y$ entries.

4.3.5 Validation of the developed algorithm: etching of spherical samples

In order to validate the proposed algorithm, the etching processes with several etchant solutions of silicon/quartz spheres are simulated, exposing all the crystallographic orientations of the substrate to the etchant. Thus, the resulting sphere can be measured before and after the etching process to calculate the actual etch rate distribution that is producing the SFM. This simulation process is commonly used to validate the calibration process of the CCA [180, 222].

For these simulations, a regular mesh of 300x300x300 points is built. The radius of the initial sphere is $\epsilon = 146\Delta x$, which represents a spherical sample of 22 mm. Using a sphere as initial surface requires using a different procedure than algorithm 14. In its place, those points that are located at ϵ distance from the center of the grid are tagged as initial surface. For this initial spherical surface determination, algorithm 16 is used, which is equivalent to algorithm 10 utilized to determine the initial surface for image reconstruction.

Once initial surface points have been determined, initial surface needs to be embedded inside a SDF. To accomplish this, the propagating algorithm 9 is used to

Algorithm 16: Determination of spherical initial surface.

```
1 Tag every mesh point as *interior* but the two most exterior rectangular cuboids that
  are tagged as *exterior*.
2 Initialize stack $P$ and include every *exterior* point.
  while $P$ is not empty do
3     Extract the first $p$ point from $P$.
      for each neighbouring point $np$ of $p$ do
          if $np$ is interior then
              if distance between $np$ and grid center $\geq \epsilon$ then
4                 Tag $np$ as *exterior*.
5                 Add $np$ to stack $P$.
              else
6                 Tag $np$ as *initial surface*.
```

calculate the corresponding distance values to initial sphere for every mesh point. Then, these values are signed as positive/negative regarding the mesh point tags *inside/outside* produced by algorithm 16. After this, the SFM lists are initialized according to the distance values of grid points.

The goal of these simulations is to measure the removed material in a certain period of time for all the 8100 directions used by experimental etch rate distributions. Accordingly, after determining the initial sphere, the exact radius for each direction must be calculated. This is performed with the algorithm 17, which calculates the radius of the sphere for all the directions. Notice that two different points of the sphere surface can produce the same spherical coordinates since a discretization of 2 deg is used for experimental etch rate distributions. As a consequence, the average radius value of all those points with the same spherical coordinates is taken, producing $Rd_{ini}(\theta, \Phi)$.

Nevertheless, due to grid resolution, it is possible that no sphere point corresponds with some location (θ, Φ), i.e. some locations $Rd_{ini}(\theta, \Phi)$ can result without any stored value. To solve this, the normal vector of those empty locations is calculated. Then, similar vectors are searched through all the sphere points and $Rd_{ini}(\theta, \Phi)$ is calculated as the average of all those radius values. Since only a hemisphere is considered for representing the etch rate distributions because of symmetry reasons, normal vectors with a negative z component are mapped to the positive hemisphere. The whole procedure is explained in algorithm 17. Notice that for initial sphere, all the L_0 points have a zero distance value.

After the initial sphere has been measured for every location (θ, Φ), the etching process is started with the desired etching conditions for 960 minutes according to evolution SFM algorithm 15. Once the sample has been etched, the radii are

Algorithm 17: Calculation of sphere radii.

```
Data: Center of the sphere: $\vec{x}_C = (x_C, y_C, z_C)$
Data: $tol = 0.01$
for every point $(x_i, y_i, z_i)$ in $L_0$ do
1     Calculate the distance to the center $r = \sqrt{(x_i - x_C)^2 + (y_i - y_C)^2 + (z_i - z_C)^2}$.
2     Calculate normalized distances to the center $d_q = \frac{q_i - q_C}{r}$ for each dimension
      $q = x, y, z$.
3     Obtain corresponding sphere coordinates $(\theta, \Phi)$ to $(d_x, d_y, d_z)$ with (4.4).
4     Consider the distance value $r = r + \phi(x_i, y_i, z_i)$.
5     Update the average distance $r_{av}$ of all the points with the same coordinates $(\theta, \Phi)$
      and store it in $Rd(\theta, \Phi) = r_{av}$.
for every empty location $Rd_{ini}(\theta, \Phi)$ do
6     Determine corresponding normal vector components:
7     $N_x = \sin\theta \cdot \cos\Phi$.
8     $N_y = \sin\theta \cdot \sin\Phi$.
9     $N_x = \sqrt{1 - N_x^2 - N_y^2}$.
      for every point $(x_i, y_i, z_i)$ in $L_0$ do
10        Calculate normalized normal vector $(d_x, d_y, d_z)$ of $(x_i, y_i, z_i)$.
          if $|N_x - d_x| < tol$ and $|N_y - d_y| < tol$ and $|N_z - d_z| < tol$ then
11            Obtain the radius $r = \sqrt{(x_i - x_C)^2 + (y_i - y_C)^2 + (z_i - z_C)^2}$.
12            Consider the distance value $r = r + \phi(x_i, y_i, z_i)$.
13            Update the average distance $r_{av}$ of all the similar normal vectors and store
              it in $Rd(\theta, \Phi) = r_{av}$.
```

measured again for every spherical location following algorithm 17. Notice that not only the distance of grid points to the center are considered but the ϕ value of each voxel, i.e. since interior points have positive values, if a L_0 point has a positive ϕ value means that the actual surface is such distance further to the center. Similarly, a negative ϕ value indicates that the surface is closer to the center. After this, $Rd_{etched}(\theta, \Phi)$ is produced, storing all the average radii for every spherical coordinates.

Accordingly, the etch rate corresponding to each location (θ, Φ) is obtained with:

$$R_{SFM} = \frac{Rd_{ini} - Rd_{etched}}{960}. \tag{4.26}$$

If this procedure is applied to silicon spheres etched with KOH 24 wt% at 70 °C, KOH 24 wt% with IPA (1 cm saturated) at 65 °C, TMAH 20 wt% at 60 °C, an isotropic etchant with $R = 1$ μm/min, KOH 40 wt% at 70 °C, TMAH 25 wt% at 80 °C, TMAH 25 wt%+ Triton 0.1 v/v at 80 °C, and a quartz sphere etched with saturated NH_4HF_2 at 70 °C, the etch rate distributions presented in Fig. 4.7 are obtained. For comparison, experimental and CCA distributions are also presented. For the CCA, only hemispheres are considered by Intellietch simulator, which is sufficient to obtain a complete etch rate distribution. In addition, silicon spheres are formed by 181x256 unit cells (a total of 259516 surface atoms) whereas quartz spheres are formed by 256x211 unit cells (472298 surface atoms).

On the other hand, for the SFM simulations a 300x300x300 grid has been employed, which correspond to an initial surface of 221546 voxels. In addition, the parameter α_F has been selected as follows: 0.48 for KOH-based experiments, i.e. (a), (b) and (e), 0.45 for TMAH-based, i.e. (c), (f) and (g), and 0.5 for quartz substrate (h).

In comparison to the experimental distributions, the LS simulations describe accurately the etch rate anisotropy for all eight cases, confirming the correctness of the developed algorithm and validating the simulation of dramatically different etchants with the proposed SFM algorithm 15. In comparison, the CCA results seem slightly noisier in all cases, but especially for the isotropic etchant and the etching solution applied to quartz. The crystallographic structure of quartz is more complicated than that of silicon, leading to a more complex atom removal sequence [180]. This results in a more complex surface morphology for the quartz sphere and, thus, the etch rate distribution becomes noisier when sampling the advancement of the etch front over various locations on the sphere.

It is also interesting to visualize the actual shape evolution of the simulated spheres for very long etch times. Fig. 4.8 shows the etched spheres for some of the etchant conditions of Fig. 4.7, simulated with the SFM developed implementation. As expected, the lower the local etch rate the sharper the corresponding protruding region, which reflects the underlying symmetry of the etch rate: two-, four- and six/three-fold around the {110}, {100} and {111} orientations, respectively. Similarly, the shape of the resulting quartz sphere follows the etch rate distribution shown in Fig. 4.7(h), producing three distinguished less-etched parts.

4.3.6 Simulation of complex MEMS

After the algorithm for simple wet etching process was validated, the simulation of more complex micromachining processes was implemented, including:

- Etching of double-sided wafers.
- Successive etching steps with the possibility of using different mask patterns and/or applying different etchants.
- Different operations to be applied on masks (top and/or bottom if double-sided substrate), including mask removal, mask deposition on non-trivial (etched) substrates, and mask removal of specific regions.

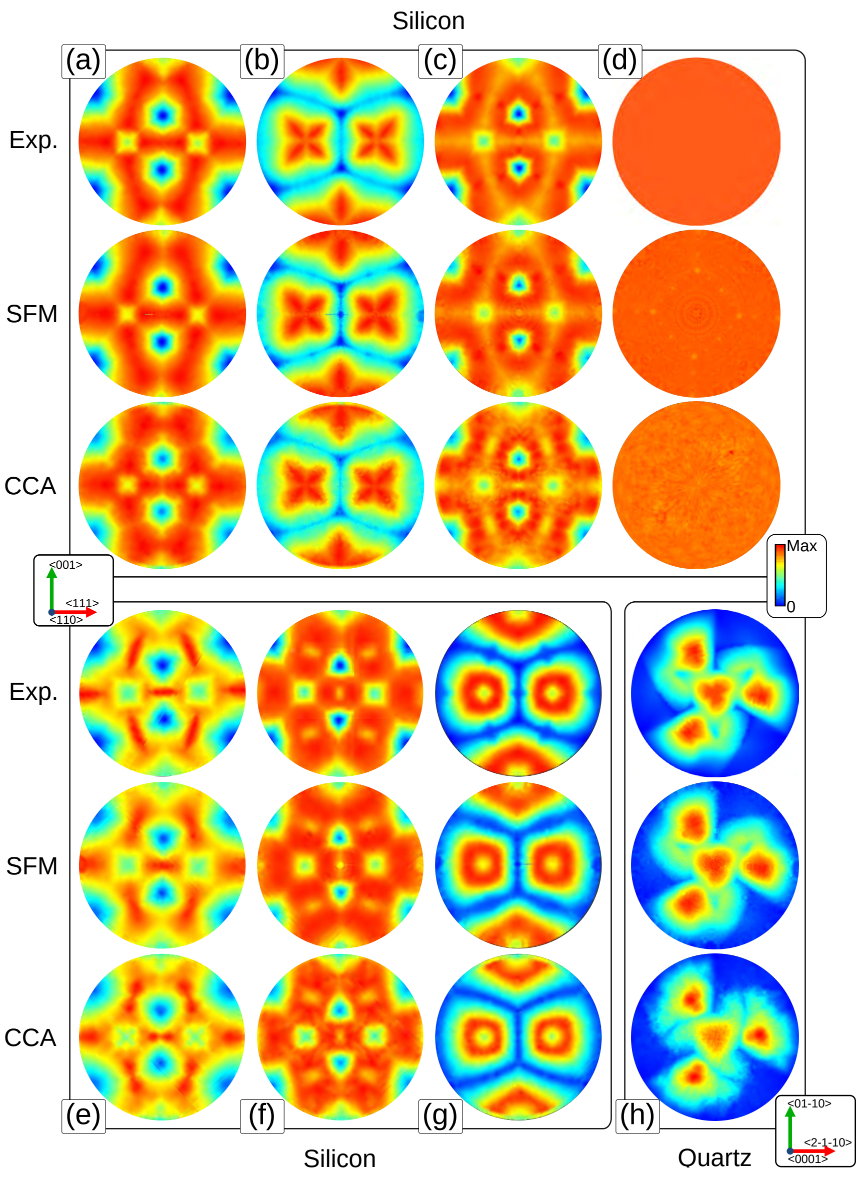


Figure 4.7: Comparison between experimental and simulated etch rate distributions using the LS and CCA methods: (a) KOH 24 wt% at 70 °C, (b) KOH 24 wt% with IPA (1 cm saturated) at 65 °C, (c) TMAH 20 wt% at 60 °C, (d) an isotropic etchant, (e) KOH 40 wt% at 70 °C, (f) TMAH 25 wt% at 80 °C, (g) TMAH 25 wt%+ Triton 0.1 v/v at 80 °C, and (h) saturated NH_4HF_2 at 70 °C. (a)-(g) Silicon substrate. (h) Quartz substrate. Adapted by author from [19] and [20].

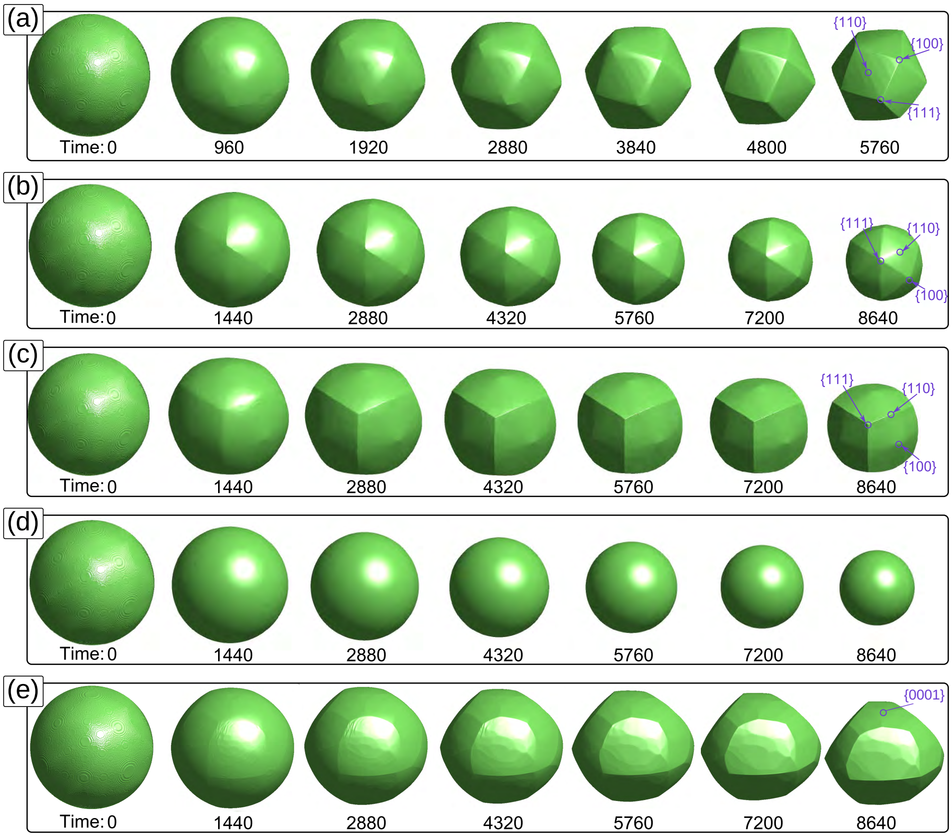


Figure 4.8: SFM-simulated evolution of a typical sphere sample of 22 mm of radius etched in: (a) KOH 40 wt% at 70 °C, (b) TMAH 25 wt% at 80 °C, (c) TMAH 25 *wt*%+ Triton 0.1 v/v at 80 °C, (d) an isotropic etchant, and (e) saturated NH_4HF_2 at 70 °C. (a)-(d) Silicon substrate. (e) Quartz substrate. The numbers represent the etch time in minutes. Adapted by author from [20].

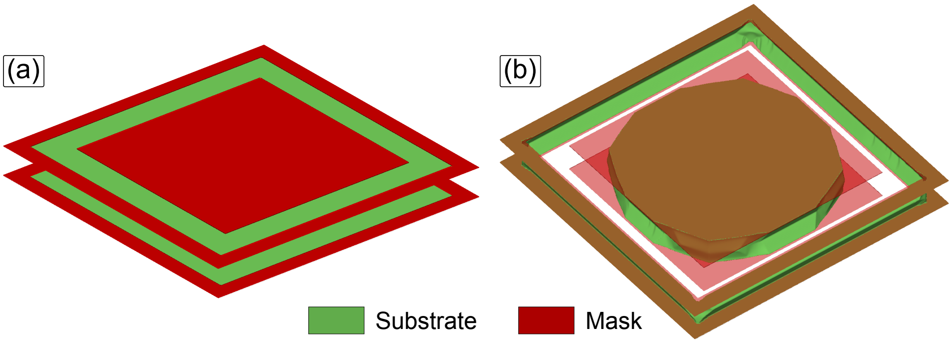


Figure 4.9: Etching example simulation of a double-sided silicon wafer: (a) substrate showing the top and the bottom initial surfaces, (b) resulting structure after the interaction of both surfaces according to etching process. In both cases, substrate surfaces and applied masks are represented.

4.3.6.1 Etching of double-sided wafers

Etching wafer on top and bottom sides simultaneously enables the fabrication of complex MEMS structures such as microprobes or accelerometers [141, 142, 173, 284]. Hence, simulation of such complex experiments is essential for MEMS design. A simple double-sided etching example is shown in Fig. 4.9.

This type of processes can be understood as two moving surfaces (top and bottom) that interact with each other to form the resulting final structure. Such interactions usually produce the vanishing of some parts of the substrate, i.e. the etching through the whole wafer. Accordingly, the LS method can handle without any additional programming effort, coalescing or disjoint of multiple surfaces since the actual surface is embedded in a higher-dimensional function.

Regarding the SFM, both surfaces can be treated as only one, i.e. the grid points of both surfaces are included in the same SFM lists. For double-sided wafer simulations, the LS grid is completely defined by user, i.e. substrate size in vertical dimension $subs_z$ is also an input parameter and the number of points in this dimension is simply calculated by:

$$num_z = \frac{num_x}{subs_x} subs_z, \tag{4.27}$$

contrary to simple etching processes where this parameter was calculated depending on the selected etchant with (4.22). Additionally, two initial surfaces must be defined at top and bottom of the grid. The top initial surface is equivalent to that of the simple etching process and corresponds to the points (i, j, k) located at plane $k = num_z - 2$. On the other hand, the bottom initial surface is the plane located at $k = 1$ (notice that indices start at 0).

Algorithm 18: Initial bottom surface determination.

```
for each grid point $(i, j, k = 1)$ do
1     $\phi(i, j, k = 1) = 0$.
2     $state(i, j, k = 1) = 0$.
      if $mask_b(i, j) = 0$ then
          add $(i, j, k = 1)$ to $L_0$.
          add $(i, j, k = 0)$ to $L_{-1}$.
          add $(i, j, k = 2)$ to $L_{+1}$.
3         $\phi(i, j, k = 0) = -\Delta x$.
4         $state(i, j, k = 0) = -1$.
5         $\phi(i, j, k = 2) = \Delta x$.
6         $state(i, j, k = 2) = 1$.
      else
7         $\phi(i, j, k = 0) = -1.5\Delta x$.
8         $state(i, j, k = 0) = -2$.
9         $\phi(i, j, k = 2) = 1.5\Delta x$.
10        $state(i, j, k = 2) = 2$.
```

Initial surfaces need to be determined according to the masks. Thus, two masks need to be loaded, namely: the mask to be applied on the top of the substrate ($mask_t$) and the mask to be deposited on the bottom of the substrate ($mask_b$). The top mask is equivalent to the mask of a simple etching process. Similarly, the same rules are applied to embed the bottom surface and to initialize SFM lists according to $mask_b$, i.e. substrate points are considered with positive distance values whereas etchant points are considered negative. Hence, algorithm 14 used for determining simple initial surfaces is applied according to $mask_t$. After that, the procedure presented in algorithm 18 is used for determining the bottom initial surface.

A two-dimensional example of a double-sided wafer is depicted in Fig. 4.10. Notice that one of the main advantages of the SFM implementation is the trivial handling of the two surfaces. The points that form both surfaces are included in the same SFM lists and the same evolution algorithm 15 is applied.

4.3.6.2 Consecutive etching processes

Another feature of the SFM-based simulator implemented in this thesis is the possibility to perform consecutive etching changing the mask patterns applied to substrate. Many MEMS structures are micromachined by applying/removing different mask patterns and etching with various solutions. Therefore, the simulation of these processes increases the usefulness of the simulator in order to design complex MEMS devices.

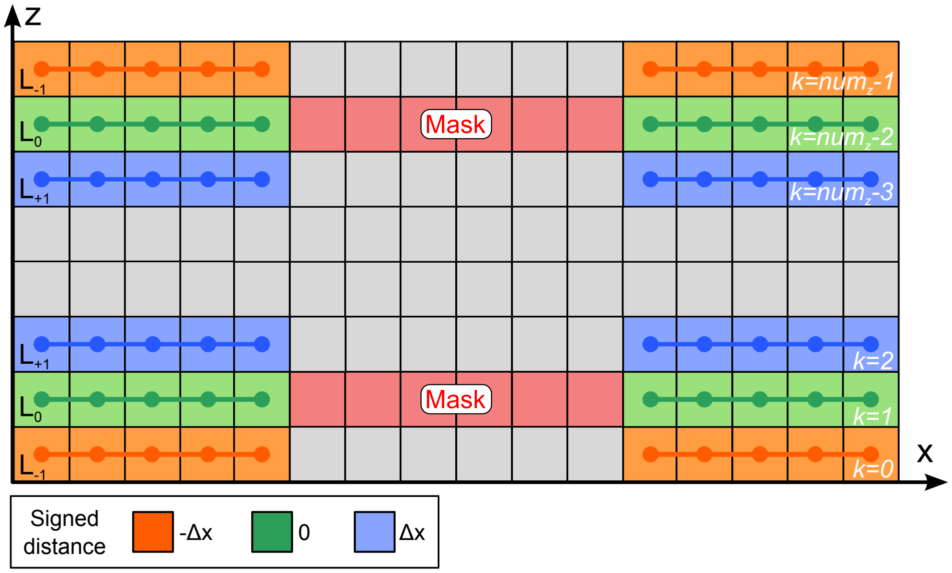


Figure 4.10: Two dimensional example of initial surfaces of a double-sided wafer partially covered with a top and a bottom masks. Voxels of both fronts are included in the same corresponding SFM lists, namely: orange, green and blue voxels represent the L_{-1}, L_0 and L_{+1} lists, respectively. Red and gray voxels are not included in any list. Positive/negative distances are associated to wafer/etchant points.

When simulating a double-sided etching, the substrate size is fixed and introduced by the user. Thus, the grid size remains constant through all the consecutive etching processes. Nevertheless, when simulating simple etching processes, vertical dimension of grid is calculated with (4.22), depending on time of etching and etch rate distribution. Thus, in the consecutive etching steps, the grid vertical dimension must be increased according to the new time of etching and the etchant selected. If the grid vertical size num_z of the first etching step was calculated with (4.22), then for an additional time of etching T_{e2} and a new etch rate distribution $R_2(\theta, \Phi)$, the new vertical dimension is obtained with

$$num_{z2} = num_z + R_2(\theta = 0, \Phi = 0) \cdot T_{e2}. \tag{4.28}$$

Then, all the variables of table 4.1 must be redefined again with this new size but keeping the previous values such that a new etching loop can be started directly.

Additionally, after each etching process, the next mask operations can be performed on both substrate sides (in double-sided simulations) by the developed simulator:

- Apply mask: the corresponding substrate side (top or bottom) will be covered according to the loaded mask pattern. If a point is already masked it will remain masked.
- Remove existing mask: the mask of the corresponding substrate side will be completely removed.
- Remove pattern: a specific substrate region will be unmasked according to the loaded pattern.

All these operations can be applied independently on top and bottom sides of substrate if it is a double-sided wafer. After the first etching process has been performed, the user can apply a combination of any of these operations before the next etching process is started. The application of a mask as well as the removing of a pattern will ask the user to load an image file. On the other hand, the mask removing operation will simple unmask every point of the desired side (in case of double sided-wafers).

Algorithm 19: Grid point status determination for mask/unmask operations.

```
Data: Grid point $(i, j, k)$
1 $uppers = false$
2 $lowers = false$
  for every point $(i, j, k_2)$, such that $k_2 > k$ do
      if $state(i, j, k_2) = 0$ then
3         $uppers = true$; break;
  for every point $(i, j, k_2)$, such that $k_2 < k$ do
      if $state(i, j, k_2) = 0$ then
4         $lowers = true$; break;
5 if $uppers = false$ and $lowers = true$ then $(i, j, k)$ is top.
6 if $uppers = true$ and $lowers = false$ then $(i, j, k)$ is bottom.
7 if $uppers = false$ and $lowers = false$ then $(i, j, k)$ is isolated.
8 if $uppers = true$ and $lowers = true$ then $(i, j, k)$ is intermediate.
```

In order to apply properly these operations to only those points of the corresponding side, the algorithm 19 has been implemented. This algorithm determines, according to the state of the points of the same column, if a grid point is:

- *top*: there are surface points only below it. It will be affected by top operations.
- *bottom*: there are surface points only over it. It will be affected by bottom operations.

- *isolated*: there are no surface points over neither below it. It will be affected by both top and bottom operations.
- *intermediate*: there are surface points below and over it. It will be affected only by those operations applied to the closest side.

Thus, depending on the tag provided by algorithm 19, mask operations can be applied correctly to the corresponding points.

Notice that, although input patterns are two-dimensional, already etched surfaces are defined over the whole three-dimensional grid. Accordingly, three main variables are defined: $mask_t$, $mask_b$ and $mask3D$. The first two are only two-dimensional variables of $num_x \cdot num_y$ grid points used for storing the patterns loaded by the user, whereas $mask3D$ is a three-dimensional variable of $num_x \cdot num_y \cdot num_z$ size which indicates for every grid point whether is masked or not.

For example, if a top loaded mask indicates that a specific point must be masked ($mask_t(i,j) = 1$), then, the *top* point of the surface in the column (i,j) must be masked. The rest of operations and point cases can be described similarly. Given top and bottom masks, $mask_t$ and $mask_b$, introduced by user, and the variable $mask3D$ that contains the mask information of the previous etching step, algorithms 20, 21 and 22 describe the operations of applying mask, removing pattern, and removing mask respectively. Notice that, when $mask_t$ and $mask_b$ are used for removing a pattern, 0 values indicate that such point must be unmasked whereas 1 values are used for denoting those surface points that must be kept intact.

Algorithm 20: Operation of applying a mask.

```
Data: $mask_t$ and $mask_b$ introduced by user.
Data: $mask3D$ from previous etching process.
Data: side = $\{top, bottom\}$ chosen by user.
for every grid point $(i,j,k)$ do
    if $state(i,j,k) = 0$ then
        if side = $top$ then $mask = mask_t$
        else if side = $bottom$ then $mask = mask_b$
        if $mask(i,j) = 1$ then
            Apply algorithm 19 to determine the $status$ of $(i,j,k)$.
            if $status$ =side or $status = isolated$ then
                $mask3D(i,j,k) = 1$
            else if $status = intermediate$ then
                if $(i,j,k)$ is closer to side surface or equidistant then
                    $mask3D(i,j,k) = 0$
```

Algorithm 21: Operation of removing a pattern.

```
Data: $mask_t$ and $mask_b$ patterns introduced by user.
Data: $mask3D$ from previous etching process.
Data: side = $\{top, bottom\}$ chosen by user.
for every grid point $(i, j, k)$ do
1   if side = $top$ then $mask = mask_t$
2   else if side = $bottom$ then $mask = mask_b$
    if $mask(i, j) = 0$ then
3       Apply algorithm 19 to determine the $status$ of $(i, j, k)$.
        if $status$ =side or $status$ = $isolated$ then
4           $mask3D(i, j, k) = 0$
        else if $status$ = $intermediate$ then
            if $(i, j, k)$ is closer to side surface then
5               $mask3D(i, j, k) = 0$
```

Algorithm 22: Operation of removing a mask.

```
Data: $mask3D$ from previous etching process.
Data: side = $\{top, bottom\}$ chosen by user.
for every grid point $(i, j, k)$ do
1   Apply algorithm 19 to determine the $status$ of $(i, j, k)$.
    if $status$ =side or $status$ = $isolated$ then
2       $mask3D(i, j, k) = 0$
    else if $status$ = $intermediate$ then
        if $(i, j, k)$ is closer to side surface then
3           $mask3D(i, j, k) = 0$
```

These algorithms are described for double-sided wafers, however, for simple processes the same algorithms can be applied but only considering top mask operations. All the simulated experiments have emulated correctly the experimental results by using these algorithms.

Once all the desired operation have been applied, the next etching process can start. Then, it is necessary to define the new SFM lists regarding these operations. Notice that each of these operations has been reflected on the variable $mask3D$, thus, new $state$ values can be assigned to grid points as well as they can be included in the corresponding SFM lists regarding their ϕ values and the value of $mask3D$ for every point.

In order to summarize and visualize properly all the operations and different etching processes that can be simulated, a workflow diagram of the whole simulator is depicted in Fig. 4.11.

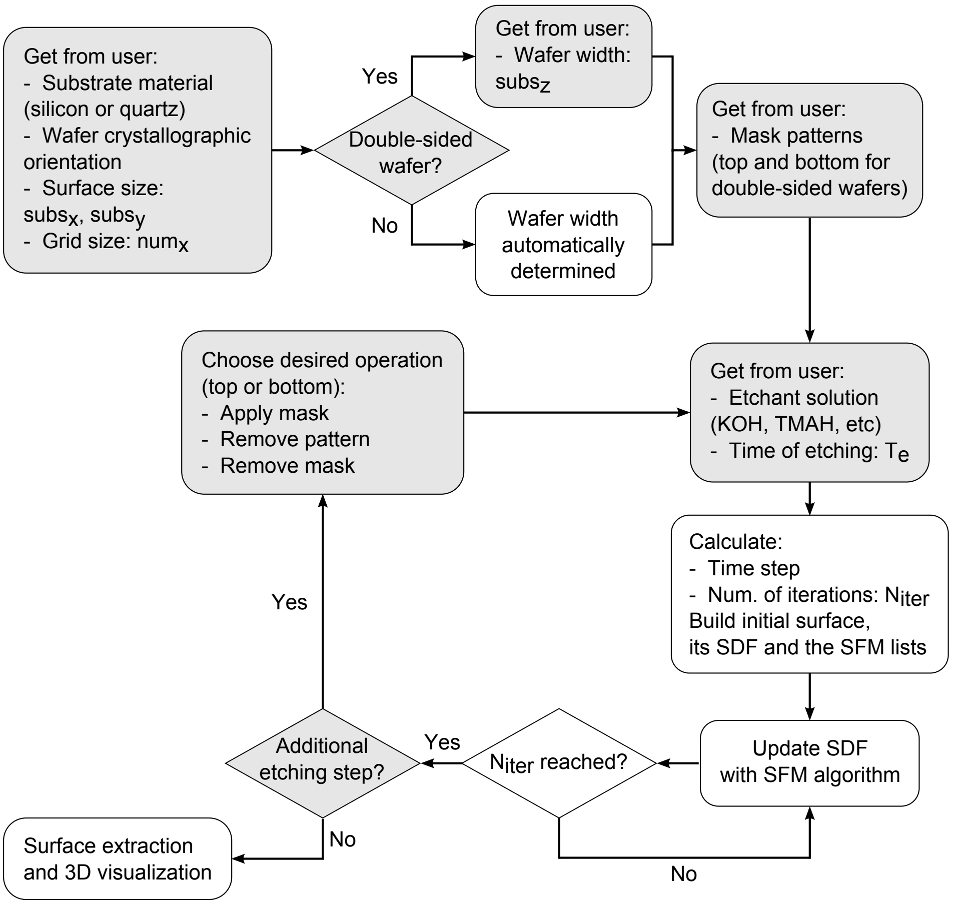

Get from user:
- Substrate material (silicon or quartz)
- Wafer crystallographic orientation
- Surface size: $subs_x$, $subs_y$
- Grid size: num_x
Double-sided wafer?
Yes
No
Get from user:
- Wafer width: $subs_z$
Wafer width automatically determined
Get from user:
- Mask patterns (top and bottom for double-sided wafers)
Get from user:
- Etchant solution (KOH, TMAH, etc)
- Time of etching: T_e
Choose desired operation (top or bottom):
- Apply mask
- Remove pattern
- Remove mask
Calculate:
- Time step
- Num. of iterations: N_{iter}
Build initial surface, its SDF and the SFM lists
Update SDF with SFM algorithm
N_{iter} reached?
Yes
No
Additional etching step?
Yes
No
Surface extraction and 3D visualization

Figure 4.11: Workflow diagram of the developed SFM-based algorithm for complex wet etching processes simulation. Gray boxes are used for operations that require user interaction, whereas white boxes are performed automatically by the program.

4.4 SFM parallel implementations

After in section 4.3 the proposed SFM was proved to be capable of correctly simulating wet etching processes and a set of algorithms were developed in order to enable the simulation of complex micromachining processes, a further optimization has been developed. In particular, two parallel algorithms have been implemented, a CPU- and a GPU-based implementations, which take advantage of the parallel intrinsic nature of the LS method since every grid point can be updated simultaneously.

The SFM relies on linked lists to track the active points of the moving surface. These data structures are not easily adapted to run on a parallel environment, thus, only a few parallel GPU-based implementations have been recently published and all of them in the field of medical image segmentation [12, 285–287]. Galluzzo et al. proposed an implementation which stores the whole grid and emulates the SFM lists with one-dimensional arrays [287]. On the other hand, approaches that reduce memory storage by splitting the grid into blocks such that, only those blocks close to the front are stored in GPU memory have been presented [285, 286]. Nevertheless, these implementations usually increase the communication between CPU and GPU, which can increase computational time.

Moreover, different numerical schemes, variables and data treatments are used for wet etching simulation with respect to medical image segmentation algorithms. To the best of author's knowledge, there is no SFM parallel implementation for wet etching simulation at the moment of writing this thesis. Accordingly, in this thesis two parallel implementations of the SFM are presented. The objective is to compare an optimized parallel multi-thread CPU implementation with an also optimized parallel many-thread GPU approach. Additionally, the GPU implementation is compared with state-of-the-art CCA GPU-based as well, in terms of execution time and accuracy of simulated results.

4.4.1 CPU implementation

The first developed parallel implementation of the SFM algorithm 15 is based on the capabilities of modern multi-core CPUs. Since the benchmarks of this study are performed on a quad-core 2.8 GHz Intel Core i7, the possibility to optimize the performance by taking advantage of specific properties of this CPU is considered. This includes the usage of Single Instruction Multiple Data (SIMD) operations, which can apply the same instruction to a multiple data objects in one clock cycle. In particular, the Streaming SIMD Extensions (SSE) operations developed by Intel are used. Based on the use of Microsoft Visual Studio 2010 in this study, the code is automatically optimized by generating SSE2 instructions using the */arch:SSE2* compilation option.

In addition, in last years the tendency of microprocessor manufacturers is to integrate multiple core in one chip instead of increasing the working frequency. There are different chooses to take advantage of these multi-core platforms, such as openMP directives [288] or the Open Source POSIX Threads (Pthreads) library recently adapted to Windows O.S. [289]. In this thesis the Pthreads library has been used due to the small size of the libraries, the efficient communications/data exchange between execution threads, and the quick learning process due to its C programming language integration. This library enables the creation of multiple execution threads that are computed simultaneously by the multiple cores of the CPU.

The main idea of the developed parallel CPU SFM is to distribute the points included in all the SFM lists among the execution threads. The way of avoiding data conflict is to enforce that each execution thread creates a sub-list of every SFM list such that, access to each list is exclusive for the thread creator. For example, if N_{th} execution threads are created such that $t_{id} = 0, \cdots, N_{th} - 1$ represents the thread identifier, the following lists are generated: $Ls_0[t_{id}], Ls_{+1}[t_{id}], Ls_{-1}[t_{id}], Ss_0[t_{id}], Ss_{+1}[t_{id}], Ss_{-1}[t_{id}]$, i.e. one sub-list of every SFM list per execution thread. Accordingly, the active points to be updated are distributed over the sub-lists, therefore, each thread computes only those points included in its own sub-list. It is important to have roughly the same amount of points in each sub-list of the same type. Otherwise, an execution thread with less points would have to wait until the rest of threads finish computing all the points. To accomplish this, points are distributed uniformly among the sub-lists, i.e. after a point has been included in a sub-list of thread $t_{id} = 0$, the next point will be included in the corresponding $t_{id} = 1$ sub-list and so on. This ensures that all the sub-lists have essentially the same amount of points. This strategy has provided the fastest simulations in all the tested experiments in comparison to other schemes, like splitting the grid in different regions such that every thread only computes those points included in its corresponding grid region.

Another important feature to consider when developing a parallel implementation is the synchronization of execution threads. This must guarantee updated values when accessing to specific stored data that can depend on other execution threads. Otherwise, wrong results would be obtained. Pthreads library provides synchronization barriers that ensures that, at the barrier point of the algorithm, all the threads have finished their previous tasks, i.e. all the threads wait until all of them reach the barrier. In the developed algorithm 15, barriers are required after steps 7, 13, 17, 18 and 19, since these steps require values stored in neighbouring positions, for instance, when updating $L_{\pm 1}$ lists, six adjacent positions of the current point are visited (see procedure 5).

Finally, in order to obtain the maximum values of viscosity factors α_q (4.15), every thread searches for the maximum values among the points included in its sub-list. Then, a search through these values must be performed. This task must be performed only by one thread, accordingly, Pthreads provides a barrier to ensure that only one thread executes a certain region of the algorithm. Hence, this serialization barrier is used before updating ϕ values (i.e. step 13) since α_q values are required in this step.

The implemented parallel CPU SFM-based algorithm for anisotropic wet etching is depicted in Fig. 4.12. In this figure, those parts of the algorithm executed in parallel by different execution threads are shown in green, whereas sequential parts are represented in white boxes and barriers and serialized sections are coloured in red. Moreover, the equivalent steps of sequential algorithm 15 are labelled in red numbers. Notice that, since the parallel CPU implementation has been developed, mainly, to compare computational efficiency with respect to the parallel GPU implementation, this implementation is only capable of simulating simple experiments, i.e. one-sided wafer substrates.

4.4.2 GPU implementation

Based on the sequential implementation of the SFM presented in algorithm 15, another parallel implementation of the SFM has been developed in this thesis, namely a CUDA C algorithm to be executed on an Nvidia GPU device. According to section 2.3.3, when executing a parallel algorithm on a GPU, several hundreds or thousands execution threads are created to simultaneously apply the same set of instructions (kernel) to different grid elements. For simplicity and clarity, only simple one-sided wafer visual examples are shown, as well as only the simple algorithm is presented. Nevertheless, the developed GPU implementation is capable of simulating complex MEMS micromachining processes (like double-sided wafers etching) as commented in section 4.3.6.

4.4.2.1 Parallelization of the SFM

A GPU implementation cannot use the same multi-core strategy of section 4.4.1 since the use of linked lists does not fit well the GPU computing philosophy. Thus, a SFM implementation without this data structure has been developed. This requires the use of an auxiliary *state* variable ($state\_aux$) to indicate the corresponding current SFM list of every grid point. Thus, the variable $state\_aux$ is updated at the same steps of algorithm 15 where points were added/removed to/from a list. Consequently, modifying the current state before required is avoided by using this auxiliary variable. The variable $state\_aux$ is a three-dimensional variable that stores the current SFM list of every grid point and it takes the values $0, 1, -1, 10, 11$, or -11 to indicate that a point is included in $L_0, L_{+1}, L_{-1}, S_0, S_{+1}$, or in S_{-1} respectively. Whereas the values 2 and -2 are used for indicating those points that are not included in any list and form part of the substrate or the etchant solution respectively. In addition, according to the SFM, only points with

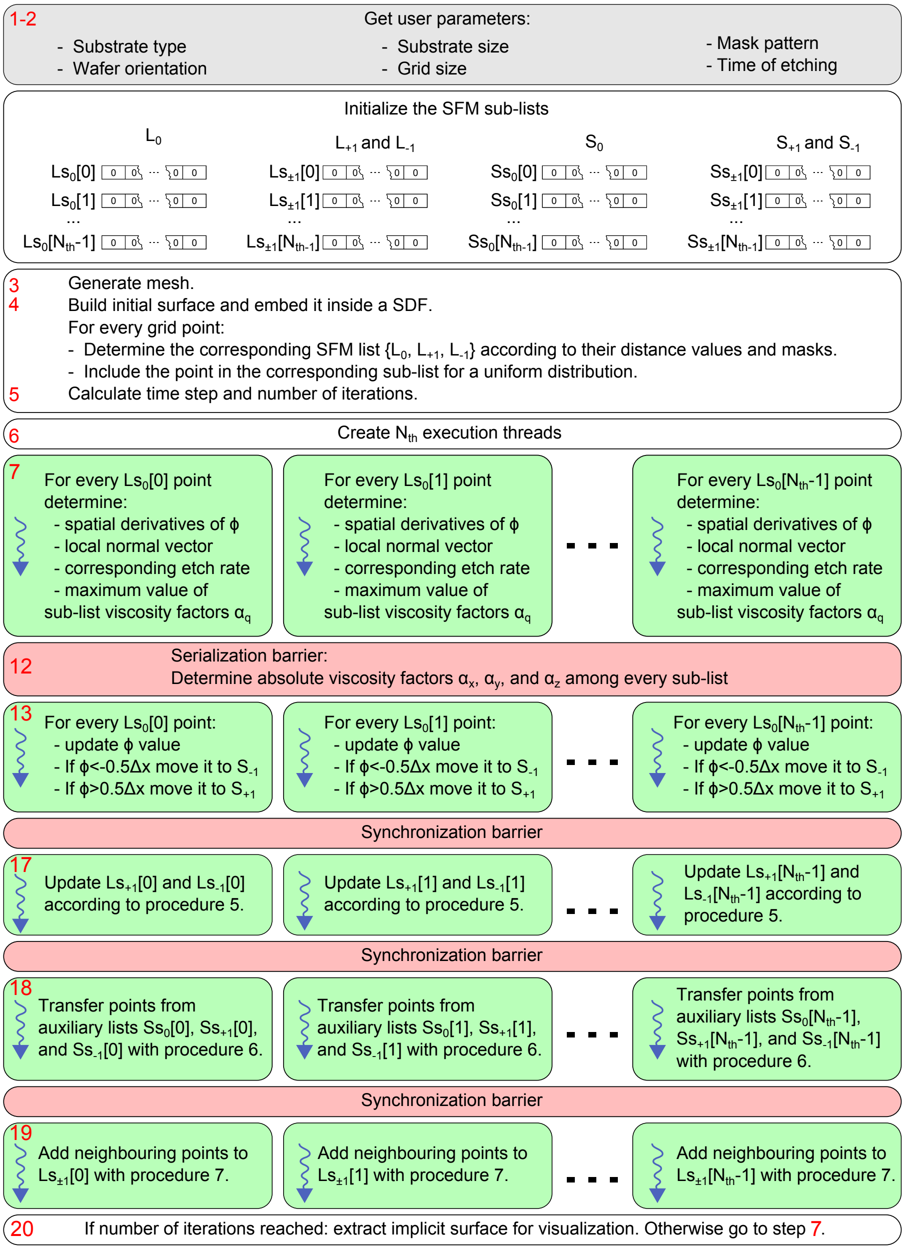

1-2
Get user parameters:
- Substrate type
- Wafer orientation
- Substrate size
- Grid size
- Mask pattern
- Time of etching
Initialize the SFM sub-lists
L_0
L_{+1} and L_{-1}
S_0
S_{+1} and S_{-1}
$Ls_0[0]$, $Ls_0[1]$, ..., $Ls_0[N_{th}-1]$
$Ls_{\pm1}[0]$, $Ls_{\pm1}[1]$, ..., $Ls_{\pm1}[N_{th}-1]$
$Ss_0[0]$, $Ss_0[1]$, ..., $Ss_0[N_{th}-1]$
$Ss_{\pm1}[0]$, $Ss_{\pm1}[1]$, ..., $Ss_{\pm1}[N_{th}-1]$
3 Generate mesh.
4 Build initial surface and embed it inside a SDF.
For every grid point:
- Determine the corresponding SFM list $\{L_0, L_{+1}, L_{-1}\}$ according to their distance values and masks.
- Include the point in the corresponding sub-list for a uniform distribution.
5 Calculate time step and number of iterations.
6 Create N_{th} execution threads
7 For every $Ls_0[0]$ point determine: - spatial derivatives of ϕ - local normal vector - corresponding etch rate - maximum value of sub-list viscosity factors α_q
For every $Ls_0[1]$ point determine: ...
For every $Ls_0[N_{th}-1]$ point determine: ...
12 Serialization barrier: Determine absolute viscosity factors α_x, α_y, and α_z among every sub-list
13 For every $Ls_0[0]$ point: - update ϕ value - If ϕ<-0.5Δx move it to S_{-1} - If ϕ>0.5Δx move it to S_{+1}
For every $Ls_0[1]$ point: ...
For every $Ls_0[N_{th}-1]$ point: ...
Synchronization barrier
17 Update $Ls_{+1}[0]$ and $Ls_{-1}[0]$ according to procedure 5.
Update $Ls_{+1}[1]$ and $Ls_{-1}[1]$ according to procedure 5.
Update $Ls_{+1}[N_{th}-1]$ and $Ls_{-1}[N_{th}-1]$ according to procedure 5.
Synchronization barrier
18 Transfer points from auxiliary lists $Ss_0[0]$, $Ss_{+1}[0]$, and $Ss_{-1}[0]$ with procedure 6.
Transfer points from auxiliary lists $Ss_0[1]$, $Ss_{+1}[1]$, and $Ss_{-1}[1]$ with procedure 6.
Transfer points from auxiliary lists $Ss_0[N_{th}-1]$, $Ss_{+1}[N_{th}-1]$, and $Ss_{-1}[N_{th}-1]$ with procedure 6.
Synchronization barrier
19 Add neighbouring points to $Ls_{\pm1}[0]$ with procedure 7.
Add neighbouring points to $Ls_{\pm1}[1]$ with procedure 7.
Add neighbouring points to $Ls_{\pm1}[N_{th}-1]$ with procedure 7.
20 If number of iterations reached: extract implicit surface for visualization. Otherwise go to step 7.

Figure 4.12: Parallel CPU SFM algorithm for wet etching simulation. Red numbers indicate the equivalent step of algorithm 15.

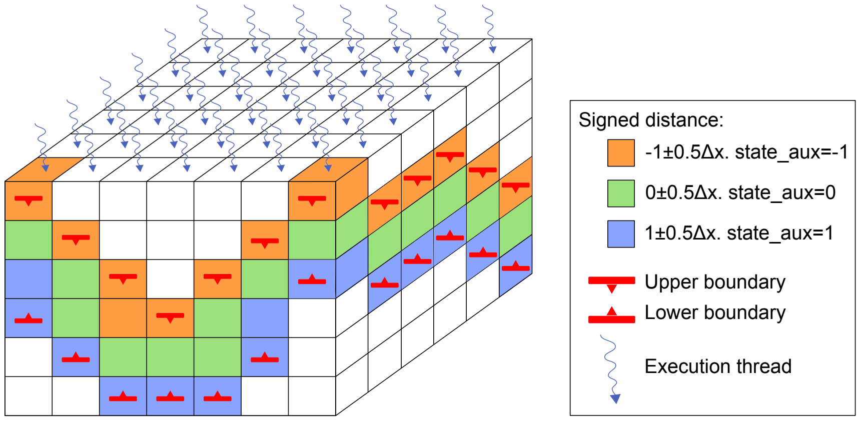


Figure 4.13: Simple three-dimensional example showing the execution thread distribution: one thread computes a vertical grid column. Additionally, *state_aux* labels and upper and lower boundaries of each thread are represented. Green voxels contain the active surface, whereas blue and orange ones corresponds with substrate and etchant solution respectively. White voxels are not computed.

$state\_aux = 0$ (or $state\_aux = \pm 1$, depending on the algorithm step) are visited and processed, whereas the points labelled as $state\_aux = \pm 2$ (i.e. no included in any list) are ignored.

A GPU can be understood as a coprocessor of the CPU which executes certain parts of the algorithm in parallel. CPU must invoke a kernel and it is in charge of allocating the necessary space memory and transferring the corresponding data in both directions, CPU to GPU and vice versa. In this study, the GPU is used for executing the etching loop of algorithm 15, similarly to the parallel CPU implementation of previous section.

The basic strategy used for the CUDA implementation, is to create as many threads as columns has the LS grid. Thus, each execution thread is in charge of applying the corresponding operations of the etching loop to every grid point contained in a specific vertical grid column, as recommended by previous CUDA optimizations for similar data access patterns [290]. Moreover, in order to mimic the SFM behaviour, each thread keeps track of the upper and lower spatial boundaries of the active region of a column, thus processing only a few voxels instead of the whole column. A three-dimensional simple example of the thread distribution is depicted in Fig. 4.13, showing the *state_aux* label of grid points as well as the column boundaries.

Likewise, it is important to pay attention on the work distribution among all the execution threads created when invoking a kernel. This implementation is executed on two different GPUs, the Nvidia GeForce GTX 260 of Tesla CUDA microarchitecture with 192 cores [110], and the Nvidia GeForce GTX 560 [113], which belongs to Fermi CUDA microarchitecture and it has 336 cores [92]. As explained in section 2.3.3, the best performance is obtained when the thread blocks size is a multiple of warp size (32 threads). It has been conclude after some tests that 256 threads per block is the optimal choice to obtain the SMs occupied enough to take advantage of the delays produced when accessing to global memory (see section 2.3.3 and Fig. 2.17). Because the work space is two-dimensional (i.e. a collection of vertical columns in three dimensions) blocks size is chosen to be 16x16 threads and the grid of blocks launched by each kernel is two-dimensional as well. The actual grid dimensions depend on the size of the LS mesh, such that the number of blocks in each dimension is obtained with:

$$\begin{aligned} Nblock_x &= \left\lfloor \frac{num_x + 15}{16} \right\rfloor \\ Nblock_y &= \left\lfloor \frac{num_y + 15}{16} \right\rfloor . \end{aligned} \tag{4.29}$$

Thus, a two-dimensional grid of blocks is created and each of these blocks contain 256 (16x16) execution threads, guaranteeing the execution of one thread per column. Notice it is possible that more threads than grid columns are created depending on grid dimensions. In such cases, the excess threads are simply not computed. The CUDA hierarchy can be visualized in Fig. 2.16.

4.4.2.2 Main variables

The variables used in the GPU SFM implementation are essentially the same described for the sequential implementation in table 4.1. Nevertheless, it is important to decide in which GPU memory must be stored each variable. Usually, accessing to data stored in global memory of GPU device takes a relevant part of computing time [291]. Hence, a very important aspect of any CUDA implementation is the need to optimize the memory bandwidth by performing memory coalescing. A coalesced access to global memory is achieved when threads of the same warp access to adjacent locations of the global memory and some conditions (which depend on the device microarchitecture) of the size of the data accessed and the memory addresses, are fulfilled [92]. A coalesced access enables to unify the memory accesses of every thread in just one operation, minimizing the amount of memory readings.

Due to the thread distribution implemented, this is easily obtained when updating values of ϕ since each thread only needs to access its memory position and each thread is located next to each other. However, when the threads need to access the neighbouring ϕ values to calculate spatial derivatives, memory coalescing is better achieved by directly using the texture memories of GPU since these are designed

Variable	**Type**	**Size**	**GPU memory**
ϕ	float	grid	Texture
ϕ_x^-	float	grid	Global
ϕ_x^+	float	grid	Global
ϕ_y^-	float	grid	Global
ϕ_y^+	float	grid	Global
ϕ_z^-	float	grid	Global
ϕ_z^+	float	grid	Global
H	float	grid	Global
$mask3D$	boolean	grid	Global
$state$	char	grid	Global
$state\_aux$	char	grid	Global
R	float	180x45	Texture
α_x	float	surface	Global
α_y	float	surface	Global
α_z	float	surface	Global
$boundary_{up}$	int	surface	Global
$boundary_{down}$	int	surface	Global

Table 4.2: Main variables allocated in GPU device and utilized in the SFM parallel GPU implementation. The variables with size *grid* are three-dimensional matrices of $num_x{\cdot}num_y{\cdot}num_z$ size. On the other hand, *surface* size corresponds to a two-dimensional matrix of $num_x \cdot num_y$ entries.

for exploiting memory access locality. With the chosen thread memory order, it has been found that this memory offers good performance for neighbourhood access. In addition, the values of the etch rates $R(\theta, \Phi)$ are also stored in texture memory to improve memory coalescing when calculating local velocity of the front with (4.17) and (4.19).

Accordingly, in table 4.2 are collected all the variables allocated in GPU memory, specifying in which type of memory are allocated. Notice that viscosity factors α_q are presented as two-dimensional matrices since the CPU is in charge of calculating the maximum values over the local maxima obtained by every execution thread (see next section). Moreover, the variables $boundary_{up}$ and $boundary_{down}$ are used for storing the upper and lower boundaries of each execution thread.

4.4.2.3 Workflow

Similarly to the parallel CPU implementation, synchronization is required after some steps in order to guarantee updated values. In a CUDA implementation, this task can be accomplished by grouping a set of instructions in a kernel since, when the CPU launches a kernel, the next kernel is not invoked until the previous has been finished and returned the control to the CPU, i.e. the kernel is a CPU blocking operation, which ensures the synchronization of all execution threads. Essentially, the algorithm 15 has been divided in five different kernels, namely the loop of step 7, the loop of step 13, and steps 17, 18 and 19. Those kernels are explained in algorithms 23, 24, 25, 26 and 27, respectively. Notice that the same amount of threads (i.e. one thread per grid column) is created when launching every kernel. In addition, every execution thread computes simultaneously all the instructions collected by the kernel. Moreover, kernels 4 and 5 are also employed to update the upper a lower boundaries of the threads.

Algorithm 23: Kernel 1 of GPU SFM implementation. It corresponds to step 7 of algorithm 15.

```
for every column point $\vec{x}_i$ with state_aux $= 0$ between upper and lower boundaries do
1   Spatial derivatives $\phi_q^{\pm}(\vec{x}_i)$.
2   Second-order central differences $\phi_q(\vec{x}_i) = \frac{\phi_q^+(\vec{x}_i)+\phi_q^-(\vec{x}_i)}{2}$.
    if Anisotropic etchant then
3       Determine normal components (4.5).
4       Obtain corresponding etch rate $R(\theta,\Phi)$.
5       Calculate maximum values of viscosity coefficients $\alpha_q$ among column points.
```

Algorithm 24: Kernel 2 of GPU SFM implementation. It corresponds to step 13 of algorithm 15.

```
for every column point $\vec{x}_i$ with state_aux $= 0$ between upper and lower boundaries do
1   Update $\phi(\vec{x}_i)$ using (4.23).
    if $\phi(\vec{x}_i) < -0.5\Delta x$ then
2       $state\_aux(\vec{x}_i) = -11$.
    if $\phi(\vec{x}_i) > 0.5\Delta x$ then
3       $state\_aux(\vec{x}_i) = 11$.
```

Another important aspect of a GPU implementation is to minimize the volume of data transferred between GPU and CPU devices in order to achieve an optimal behaviour [92]. Accordingly, in the developed implementation, the CPU generates the initial surface and, then, the variables $\phi, state, state\_aux, R(\theta,\Phi), boundary_{up}$ and $boundary_{down}$ are transferred to the GPU where the etching process is simulated. Once the time of etching has been reached, the final SDF ϕ is taken

Algorithm 25: Kernel 3 of GPU SFM implementation. It corresponds to step 17 of algorithm 15.

```
   for every column point $\vec{x}_i$ between upper and lower boundaries do
       if $state\_aux(\vec{x}_i) = 1$ then
1          Among the six neighbouring points of $\vec{x}_i$, find the point with minimum distance
           $\phi(\vec{x}_b)$ value and $state = 0$.
           if no point with $state = 0$ is found then
2              $state\_aux(\vec{x}_i) = 2$.
3              $state(\vec{x}_i) = 2$.
           else
4              Update $\phi(\vec{x}_i) = \phi(\vec{x}_b) + \Delta x$.
5              if $\phi(\vec{x}_i) \in [-0.5\Delta x, 0.5\Delta x]$ then $state\_aux(\vec{x}_i) = 10$.
6              if $\phi(\vec{x}_i) > 1.5\Delta x$ then $state\_aux(\vec{x}_i) = 2$.
       if $state\_aux(\vec{x}_i) = -1$ then
7          Among the six neighbouring points of $\vec{x}_i$, find the point with maximum distance
           $\phi(\vec{x}_b)$ value and $state = 0$.
           if no point with $state = 0$ is found then
8              $state\_aux(\vec{x}_i) = -2$.
9              $state(\vec{x}_i) = -2$.
           else
10             Update $\phi(\vec{x}_i) = \phi(\vec{x}_b) - \Delta x$.
11             if $\phi(\vec{x}_i) \in [-0.5\Delta x, 0.5\Delta x]$ then $state\_aux(\vec{x}_i) = 10$.
12             if $\phi(\vec{x}_i) < -1.5\Delta x$ then $state\_aux(\vec{x}_i) = -2$.
```

by CPU in order to proceed with the visualization process. On the other hand, the maximum values of viscosity factors (4.15), must be searched over all the grid points that from part of the active surface, thus, each execution thread obtains the local maximum value within its corresponding column. Then, all these local maxima (notice that this values are stored in a two-dimensional variable of the surface size $num_x \cdot num_y$) are transferred to the CPU, which has to find the absolute maximum values of the viscosity factors among all the local values found by each thread. Once obtained, the three maxima α_x, α_y, and α_z are returned to the GPU, which then continues with the next step. Although there is room for further optimization, the time required by the CPU to find the maximum value over a two-dimensional grid is negligible while the time spent doing the transfer from GPU to CPU is only about 2.5% of the total simulation time (as commented in section 4.5).

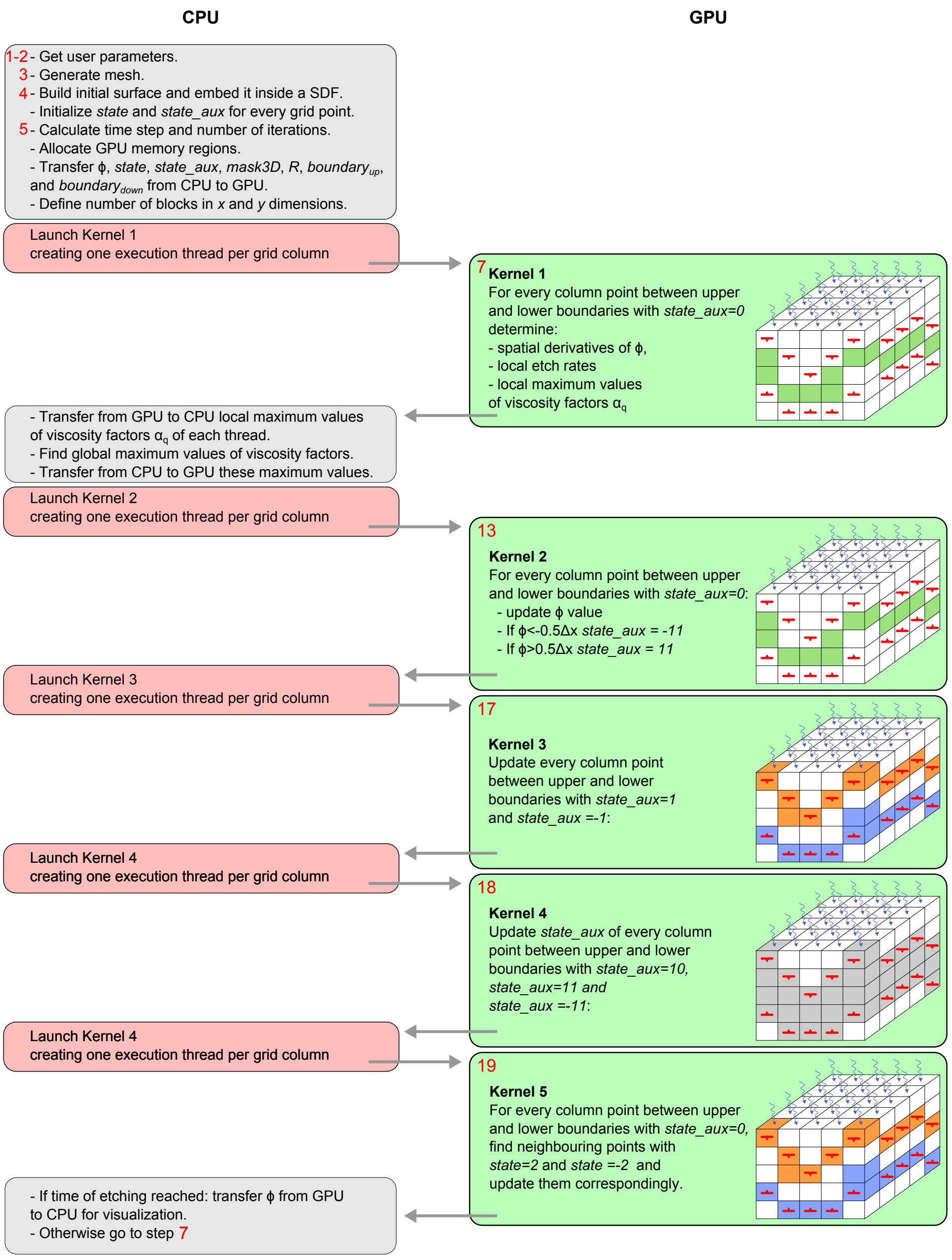
The workflow of this GPU implementation can be visualized in Fig. 4.14, including the different kernels execution and a visual representation of those affected grid points for a simple example, and the data transfers between CPU and GPU.

Algorithm 26: Kernel 4 of GPU SFM implementation. It corresponds to step 18 of algorithm 15.

```
1  Initialize $min_{aux} = num_z$ and $max_{aux} = 0$.
   for every column point $\vec{x}_i = (i, j, k)$ between upper and lower boundaries do
       if $state\_aux(\vec{x}_i) = 10$ then
2          $state(\vec{x}_i) = 0$.
3          $state\_aux(\vec{x}_i) = 0$.
       if $state\_aux(\vec{x}_i) = 11$ then
4          $state(\vec{x}_i) = 1$.
5          $state\_aux(\vec{x}_i) = 1$.
       if $state\_aux(\vec{x}_i) = -11$ then
6          $state(\vec{x}_i) = -1$.
7          $state\_aux(\vec{x}_i) = -1$.
       if $state\_aux(\vec{x}_i) = 0$ or $state\_aux(\vec{x}_i) = \pm 1$ then
8          if $min_{aux} > k$ then $min_{aux} = k$.
9          if $max_{aux} < k$ then $max_{aux} = k$.
10 if $boundary_{down}(i, j) > min_{aux}$ then $boundary_{down}(i, j) = min_{aux}$
11 if $boundary_{up}(i, j) > max_{aux}$ then $boundary_{up}(i, j) = max_{aux}$
```

Algorithm 27: Kernel 5 of GPU SFM implementation. It corresponds to step 19 of algorithm 15.

```
   for every column point $\vec{x}_i = (i, j, k)$ with $state\_aux = 0$ between upper and lower boundaries do
       for neighbouring points $\vec{x}_b = (i_b, j_b, k_b)$ with $state = 2$ do
1          $state(\vec{x}_b) = 1$.
2          $state\_aux(\vec{x}_b) = 1$.
3          $\phi(\vec{x}_b) = \phi(\vec{x}_i) + \Delta x$.
4          if $boundary_{down}(i_b, j_b) > k + 1$ then $boundary_{down}(i_b, j_b) = k + 1$
       for neighbouring points $\vec{x}_b$ with $state = -2$ do
5          $state(\vec{x}_b) = -1$.
6          $state\_aux(\vec{x}_b) = -1$.
7          $\phi(\vec{x}_b) = \phi(\vec{x}_i) - \Delta x$.
```



1-2 - Get user parameters.
3 - Generate mesh.
4 - Build initial surface and embed it inside a SDF.
- Initialize *state* and *state_aux* for every grid point.
5 - Calculate time step and number of iterations.
- Allocate GPU memory regions.
- Transfer ϕ, *state*, *state_aux*, *mask3D*, *R*, $boundary_{up}$, and $boundary_{down}$ from CPU to GPU.
- Define number of blocks in *x* and *y* dimensions.

Launch Kernel 1
creating one execution thread per grid column

7 **Kernel 1**
For every column point between upper and lower boundaries with *state_aux=0* determine:
- spatial derivatives of ϕ,
- local etch rates
- local maximum values of viscosity factors α_q

- Transfer from GPU to CPU local maximum values of viscosity factors α_q of each thread.
- Find global maximum values of viscosity factors.
- Transfer from CPU to GPU these maximum values.

Launch Kernel 2
creating one execution thread per grid column

13
Kernel 2
For every column point between upper and lower boundaries with *state_aux=0*:
- update ϕ value
- If ϕ<-0.5Δx *state_aux = -11*
- If ϕ>0.5Δx *state_aux = 11*

Launch Kernel 3
creating one execution thread per grid column

17
Kernel 3
Update every column point between upper and lower boundaries with *state_aux=1* and *state_aux =-1*:

Launch Kernel 4
creating one execution thread per grid column

18
Kernel 4
Update *state_aux* of every column point between upper and lower boundaries with *state_aux=10, state_aux=11 and state_aux =-11*:

Launch Kernel 4
creating one execution thread per grid column

19
Kernel 5
For every column point between upper and lower boundaries with *state_aux=0*, find neighbouring points with *state=2* and *state =-2* and update them correspondingly.

- If time of etching reached: transfer ϕ from GPU to CPU for visualization.
- Otherwise go to step 7

Figure 4.14: Parallel GPU SFM algorithm for wet etching simulation. Red numbers indicate the equivalent step of algorithm 15.

4.5 Results and comparisons

In this chapter a total of three SFM-based wet etching simulator implementations has been developed, namely: (i) a pure sequential Java, (ii) a parallel CPU using Pthreads library in C programming language, and (iii) a parallel GPU implemented in CUDA C. The three implementations obtain exactly the same results for the same experiment conditions, thus, the accuracy obtained by them is the same. Accordingly, the three versions implement the SFM algorithm 15, use first-order differences (2.26), (2.27) for forward and backward derivatives, and second-order central derivatives (2.28) to calculate the normal vector components (4.5). Furthermore, $\alpha_F = 0.48$ is used for KOH-based etchants, $\alpha_F = 0.5$ is utilized for NH_4HF_2, and $\alpha_F = 0.45$ for TMAH-based solutions.

In this section several experiments with different etchant conditions are simulated. To prove the effectiveness of the developed SFM-based algorithms, every result is compared with the corresponding experimental one. In addition, every simulation result is also compared to state-of-the-art CCA results, in terms of computational efficiency and accuracy of the results.

This section is structured as follows. First, the effect of mesh resolution on the accuracy of resulting structures is studied. For the CCA approach, this effect is similar and it is also studied. Later, several simple examples are simulated with both parallel SFM implementations, the CPU and the GPU versions, to be compared in terms of computational efficiency. Following, the sequential SFM implementation is compared to a similar sequential CCA approach. The same simple experiments are simulated to obtain a comparison in terms of computational efficiency. Finally, a comparison of complex MEMS results is performed between SFM GPU implementation and a GPU-based CCA approach. These results are compared with experimental ones too.

4.5.1 Mesh resolution impact

The underlying grid in the SFM simulations effectively models the etched substrate and its resolution affects the final accuracy of results. Typically, the larger the number of grid points, the better the accuracy, a phenomenon that is also observed in atomistic models. The grid points corresponds to voxels for the SFM and to UCs for the CCA approach. Notice that, due to crystallographic structure of silicon and quartz, the number of atoms that form a UC changes depending on surface orientation and direction of the substrate cut (i.e. the rotation of the wafer).

To quantify the effect of the grid size, two experiments are considered and simulated by the developed SFM and two CCA approaches, namely: the Constant-Time-Stepping (CTS) implementation is employed since this implementation uses a constant time step like the developed SFM algorithm and, thus, it will be used in next sections for computational efficiency comparison. On the other hand, the compensated CTS implementation can achieve better accuracy since modifies the time step during the simulation according to current surface and etch rates. Both CCA approaches were presented by Ferrando et al. [292] and are currently considered the most advanced atomistic wet etching simulators. Moreover, these approaches are written in the same programming languages than those SFM developed in this thesis, i.e. Java for sequential implementations and CUDA C for the parallel versions. Hence, both approaches can be perfectly used for comparison with the SFM developed approaches.

In the first place, the underetching of the convex corners of a square-shaped mask pattern on a 128x128 μm^2 substrate etched in KOH 40 wt% at 70 °C for 30 min is simulated. For this first example, the CTS implementation of the CCA has been utilized. As shown in Fig. 4.15(a), the measured width of the obtained mesa structure decreases as the number of grid points is increased (voxels in SFM and UCs in CCA, each one formed by 4 atoms in this case), following a negative exponential behaviour that essentially saturates at the 512x512 grid size. Considering a discrepancy of 2% with respect to this value, the fitted exponentials indicate that the minimal grid resolution to achieve this error is $68,062$ voxels and $28,589$ UCs for SFM and CCA, respectively. Thus, the SFM method requires about double the number of points to reach similar accuracy as CCA. Nevertheless, the discrepancy between both approaches for 512x512 grid points is only 3.8%, thus concluding that the SFM seems suitable for the simulation of anisotropic etching of real structures. This example focuses on the effect of the mesh resolution on relative coarse features, such as the width of the resulting mesa. Accordingly, the obtained mesh resolutions will be used later for the simulation of simple experiments.

On the other hand, a rather complex dual-axis microprobe structure is now considered (the micromachining process is explained later) [173]. For this comparison, the compensated CTS-CCA is employed and the width of the beam shown in Fig. 4.15(b) is measured for several grid resolutions shown in the corresponding graph. Similar to the first example, this measurement follows a negative exponential behaviour, effectively saturating at 11.0 and 11.25 μm for SFM and CCA, respectively, for the 800x453 surface grid size. Since the discrepancy between the converged beam widths is only 0.25 μm in comparison to the large size of the substrate (4000x2266 μm^2), it is concluded that the SFM method appears to be a suitable alternative for the simulation of anisotropic etching of advanced structures. In contrast to high resolutions, where the differences between both methods are minimal, at low resolutions the discrepancy is higher. Although the SFM can reach sub-voxel precision by definition, the mask transfer does not. Thus, the difference by a single masked voxel can be significant, becoming a relevant limit when choosing the surface grid size. Considering that a 10% discrepancy with respect to the converged beam width implies a difference

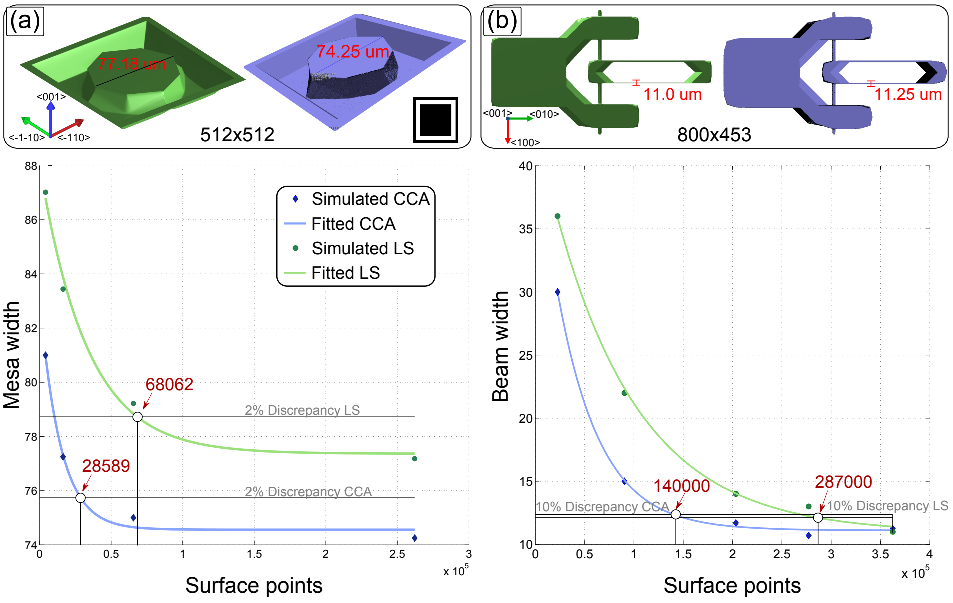


Figure 4.15: Representation of the mesh resolution effect on simulated results for both the SFM and the benchmark CCA method. In (a), the CTS-CCA implementation is used for comparison, whereas in (b) the compensated CTS-CCA is used. The minimum grid sizes required to obtain a maximum discrepancy of (a) 2% and (b) 10% with respect to the measurements at the (a) 512x512 and (b) 800x453 grids are shown, including the resulting structures at this resolutions.

of only 1.1 and 1.13 μm for SFM and CCA, respectively, the fitted exponentials indicate that the minimal grid resolution to achieve this 10% deviation is about $287,000$ voxels and $140,000$ UCs (for this particular orientation and rotation each UC contains 8 silicon atoms) for SFM and CCA, respectively. Thus, for this structure, the LS method requires about double the number of points than the CCA implementation to avoid errors due to grid resolution. Since this is the most complex three-dimensional structure considered in this thesis, this same grid resolution can be safely used to compare meaningfully the two simulation procedures for the complex structures presented in section 4.5.4.

4.5.2 Parallel CPU vs GPU

In this section both developed parallel implementations of the SFM are compared, i.e. the parallel CPU version of section 4.4.1 against the GPU implementation of section 4.4.2. The goal of this comparison is to find out which parallel environment can obtain better results in terms of execution time, when simulating wet etching processes by means of the SFM. The testing machine consists of an Intel Core i7 at 2.8 GHz with 4 GB of RAM using 64 bit Windows-based and an Nvidia GeForce GTX 560 GPU device.

For the CPU implementation it has been found that creating 8 execution threads is the optimal choice. Moreover, in the compilation of the code, the following options have been selected: /arch:SSE2, /fp:fast, /Ox and /Ot, thus ensuring an optimized and fast program. On the other hand, for the GPU version, two-dimensional blocks of 16x16 threads have been created.

The relative performance between both parallel implementations in terms of computational time and memory usage is compared. For this purpose, several microengineering structures obtained by the etching of patterned Si{100} wafers in KOH 30 wt% at 80 °C are simulated and studied [293]. The simulated etchant is actually KOH 35 wt% at 80 °C, which has a very similar etch rate distribution [191]. In addition, two simple etching processes with an isotropic etchant have been simulated.

Fig. 4.16 compares all the shapes etched with the KOH solution. From left to right, the different columns correspond to (i) the used mask patterns, (ii) the experimental results [293], (iii) the SFM-simulated fronts and (iv) the CCA-based results. The experimental and the CCA results have been included in this figure for completeness since in section 4.5.3 a comparison with a CCA approach is performed with the same experiments. On the other hand, in Fig. 4.17 the two isotropic examples are presented, column (i) shows the applied mask patterns, (ii) is the SFM results and (iii) the CCA results.

The green figures are the resulting structures for both parallel SFM implementations since both of them implement the same algorithm and obtain the same results. The grid sizes of each simulation have been chosen to satisfy the 2% error condition described in section 4.5.1 for simple experiments.

Table 4.3 collects the surface grid sizes employed, the amount of memory utilized and the simulation time of both implementations. Since this section is focused on the comparison of both parallel SFM implementations in terms of computational performance, the features of the experiment and measurements of the resulting structures are omitted and they will be discussed later when comparing the accuracy obtained against the CCA approach.

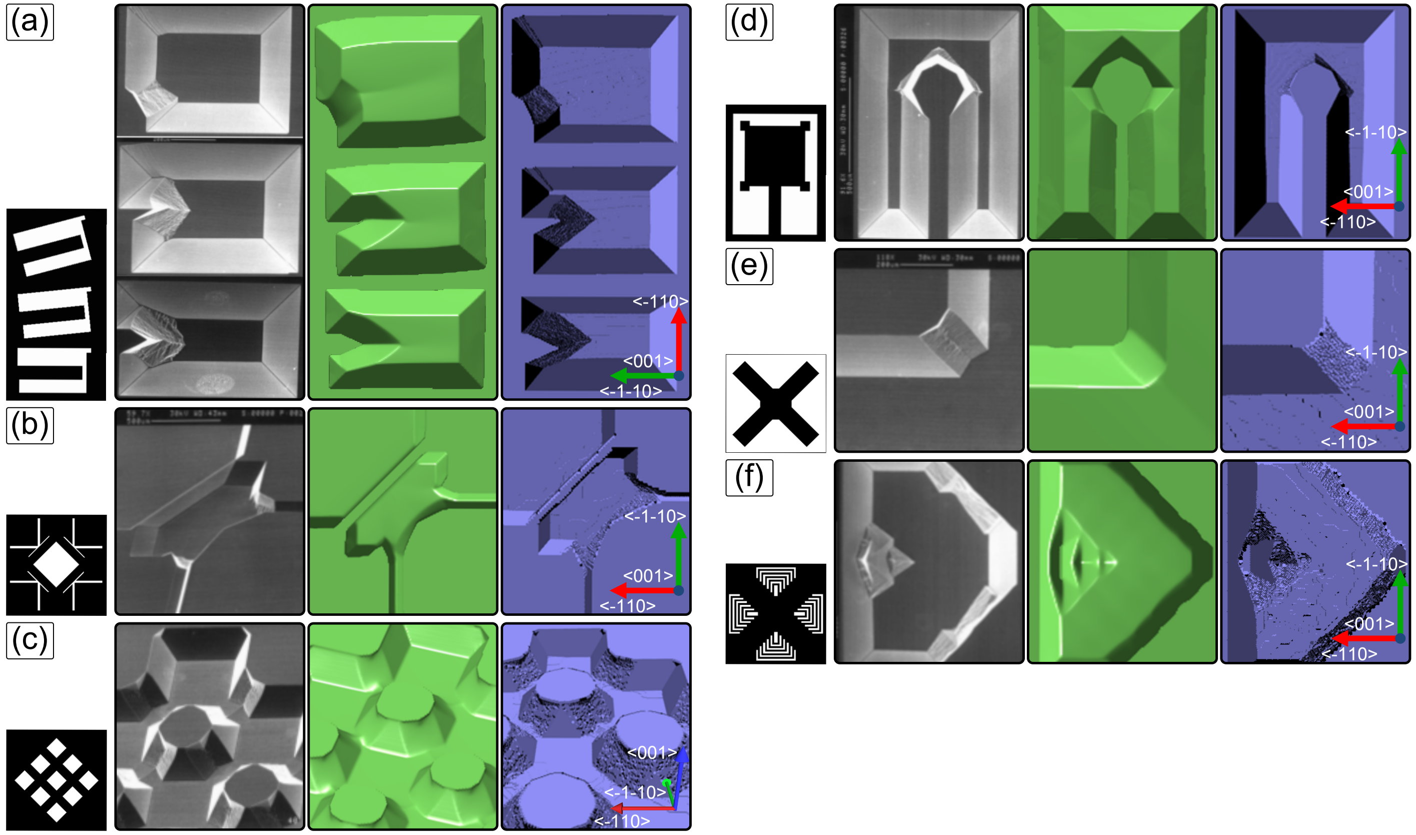


Figure 4.16: Comparison between experimental and simulated structures etched in a KOH-based solution: (left columns) applied masks, (center-left) experiments [293], (center-right) SFM results, (right) CCA results. Reproduced from [20].

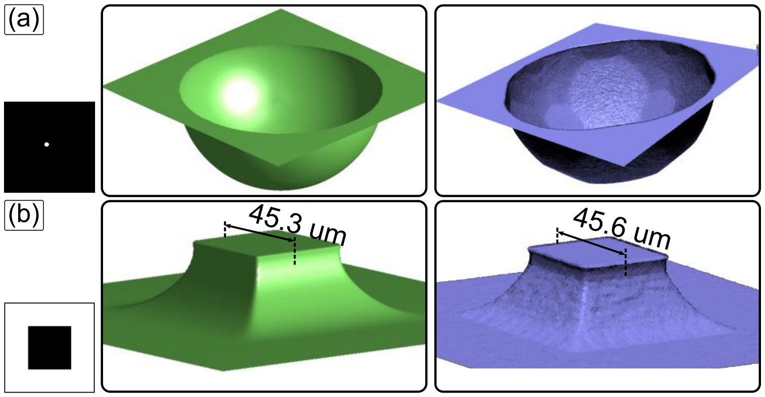


Figure 4.17: Comparison between simulated etched structures with an isotropic etchant solution: (left column) applied masks, (center) SFM results, (right) CCA results. Reproduced from [20].

Experiment	Surface grid (voxels)	Used memory (MB)			Simulation time (s)	
		CPU		GPU	CPU	GPU
		Main	Lists	Main		
Fig 4.16(a)	338x201	79.4	2.12	89.8	2.7	0.6
Fig 4.16(b)	261x262	28.9	0.9	37.9	0.37	0.15
Fig 4.16(c)	261x261	42.0	1.14	51.4	0.84	0.24
Fig 4.16(d)	304x223	105.6	2.52	116.79	4.2	0.77
Fig 4.16(e)	261x261	66.3	2.54	76.43	2.8	0.43
Fig 4.16(f)	261x261	39.8	1.13	49.14	0.77	0.22
Fig 4.17(a)	261x261	238.8	2.25	253.78	2.15	1.34
Fig 4.17(b)	261x261	95.1	2.49	105.99	1.63	0.22

Table 4.3: Details of the simulated examples for the comparison between parallel CPU and GPU implementations of the SFM.

The main memory usage has been calculated theoretically by considering the main variables shown in table 4.1 for the CPU implementation and the variables of table 4.2 for the GPU approach, and taking into account the surface sizes shown in table 4.3. On the other hand, the total number of points stored by all the SFM lists has been checked in every iteration for the whole simulations, thus, the memory utilized by the SFM lists shown in table 4.3 has been calculated by considering the maximum number of points included in all lists and taking into account that every point included in a lists contains 3 floats (the 3 spatial coordinates), i.e. a total of 12 bytes per included point. Notice that *float* and *integer* data types occupy 4 bytes, whereas *char* and *boolean* only occupy 1 byte. Moreover, the number of points in vertical dimension have been calculated with (4.22).

Regarding the used memory collected in table 4.3, the GPU version does not use any linked lists but requires storing the auxiliary state variable, the maximum viscosity factor values of each column as well as the upper and lower boundaries of each thread, implying a small increase of the main memory usage. In terms of computational performance, the table demonstrates that the GPU algorithm runs up to 7.4 times faster than the CPU version. Although the results confirm that current CPUs stand as an acceptable platform to perform this type of wet etching simulations, the massively parallel architecture of current GPUs provides better computational efficiency with only a slightly larger use of memory. In fact, the GPU version is expected to become increasingly more efficient when simulating larger substrates, since the massively parallel architecture of these devices is progressively more suitable when more and more threads are created, in contrast to the multi-core CPU. Nevertheless, both implementations of the SFM algorithm produce simulated results within a few seconds and combine both the computational efficiency and the algorithmic accuracy required for the realistic simulation of wet etching fronts for improved MEMS design.

4.5.3 Sequential CCA vs sequential SFM

In this section, the same experiments of previous section are simulated in order to compare the developed sequential SFM algorithm with a sequential CCA approach. In particular, the CTS-CCA implementation is used for comparison [292]. Both approaches are written in sequential Java and use similar data structures as well as single precision operations since it has been demonstrated that this level of precision is sufficient to produce accurate results [102]. The testing machine consists of an Intel Core i7 at 2.8 GHz with 4 GB of RAM using 64 bit Windows-based server Java Virtual Machine (version 1.7.0_03).

The purpose of this section is to compare the computational cost, memory use and accuracy of similar implementations of both methods. Accordingly, the same experiments of previous section are simulated again, producing the same results

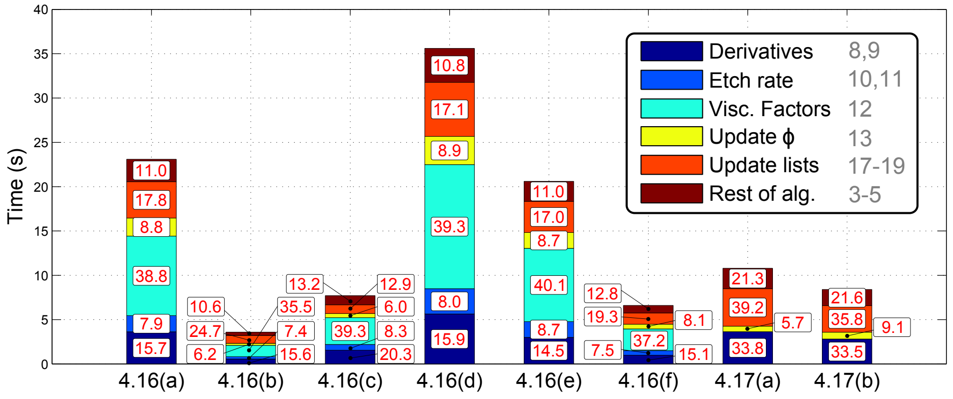


Figure 4.18: Representation of the execution times obtained with the SFM sequential Java-based algorithm for the simulation results presented in Fig. 4.16 and 4.17. The red numbers indicate the contribution of the main parts of algorithm 15. The gray numbers indicate the corresponding step of the algorithm.

presented in Fig. 4.16 and 4.17. Whereas the SFM results are presented in green, the blue structures correspond to the CCA results.

The grid sizes, chosen for each simulation to satisfy the 2% error described in the context of Fig. 4.15(a) for simple etching processes, are collected in table 4.4, which also presents some features of the experiment as well as the etched depth, memory use and computational time for both methods. In these examples, each UC of the CCA method is formed by 4 atoms.

4.5.3.1 Anisotropic results

According to Fig. 4.16, the simulated microstructures by both SFM and CCA are very similar to the experimental shapes. This is confirmed by the values of the simulated etched depths collected in table 4.4, which are in close proximity to the experimental values. These results demonstrate the reliability of the SFM simulator, achieving similar accuracy than CCA approach. It is important to stress the less noisy results produced by the SFM in comparison with the CCA model as can be observed in all the tested experiments. This is consistent with the simulated etch rates of section 4.3.5 (see Fig. 4.7).

As shown in table 4.4, the required computational times for both methods are within the same order of magnitude for all tests, although CCA can be up to 2 times faster for anisotropic etchants. In turn, the reported main memory refers to the main variables stored during the whole simulation, which directly reflects the grid size. Additionally, the memory referred as *lists* corresponds to the points

	Experiment features			Surface grid size		Depth (μm)		Used memory (MB)		Sim. time (s)	
	Substrate size (μm)	Time (min)	Depth (μm)	SFM (voxels)	CCA (UCs)	SFM	CCA	SFM Main(lists)	CCA Main(lists)	SFM	CCA
Fig 4.16(a)	2370x1416	200	225	338x201	219x130	231.4	229.6	79.4(2.12)	347.3(2.84)	23.1	14.7
Fig 4.16(b)	4720x4750	150	166	261x262	167x169	174.4	173.9	28.9(0.9)	106.4(1.15)	3.6	2.1
Fig 4.16(c)	3782x3782	200	225	261x261	169x169	232.5	233.4	42.0(1.14)	175.6(1.57)	7.7	3.8
Fig 4.16(d)	1564x1148	200	225	304x223	198x145	231.5	231.8	105.6(2.52)	481.2(3.39)	35.6	28.9
Fig 4.16(e)	2305x2305	200	225	261x261	169x169	231.0	233.9	66.3(2.54)	280.4(3.0)	20.6	20.0
Fig 4.16(f)	4170x4170	200	225	261x261	169x169	225.9	231.2	39.8(1.13)	158.6(1.49)	6.6	4.9
Fig 4.17(a)	250x250	100	–	261x261	169x169	98.3	105.7	238.8(2.25)	1144(4.23)	10.8	113.6
Fig 4.17(b)	200x200	30	–	261x261	169x169	30.0	32.0	95.1(2.49)	450(3.33)	8.4	37.5

Table 4.4: Experimental and simulation details for the structures shown in Fig. 4.16 (anisotropic etchants) and 4.17 (isotropic etchants).

stored in the SFM lists and similar structures for the CCA model. The lists use much less memory than the grid since only the necessary information to access to the main grid is stored. Although the SFM implementation typically requires a finer grid, table 4.4 shows that the CCA method needs to store more atoms, requiring between 3.7 and 4.8 times more memory.

Fig. 4.18 presents the execution time of every simulated structure of table 4.4, including the contribution of each part of the algorithm. Accordingly, the larger computational cost for SFM is assigned primarily to the need to determine the maximum value for the viscosity factors (4.15) over all active points, which represents 35% – 42% of the simulation time. Comparatively, the computation of the spatial derivatives by (2.26), (2.27), and (2.28) takes 14% – 22% of the simulation time. This involves on average more operations than the processing of the atomistic neighbourhood in the CCA. Finally, the task of updating the content of the lists (i.e. steps 17 through 19 of algorithm 15) is also relevant, representing 13% – 26% of the whole time.

4.5.3.2 Isotropic results

For isotropic etchants, the etch rate remains essentially constant along any direction. This behaviour can be tested by using a mask with a small circular opening. If the etching process is truly isotropic, a perfect hemispherical cavity will be developed into the substrate after prolonged etching. Fig. 4.17(a) compares the results of such a computational experiment for the SFM and CCA method. Both methods reproduce the expected result, nevertheless, it is concluded that the CCA results are noisier and less isotropic, which is consistent with the etch rate distribution for the isotropic etchant shown in Fig. 4.7. A similar conclusion is obtained from Fig. 4.17(b), where a square mask pattern leads to the formation of a mesa structure. The measurements of both experiments obtained by the two methods are very similar, concluding that the SFM is capable of simulating properly isotropic etchants.

An interesting feature of the developed SFM algorithm applied to isotropic etching is that it is not necessary to search the values for the artificial viscosity of (4.15) since etch rates are always constant and, thus, $\alpha_q = 1$ for all spatial dimensions as commented in section 4.2.1. The spherical coordinates of the normal vector are not required either as reflected in algorithm 15, thus resulting in a faster algorithm for isotropic etchants. This is reflected in the simulation times shown in table 4.4, where SFM is 10.5 and 4.5 times faster than CCA, respectively, while the memory used by SFM remains lower, as for the anisotropic examples.

According to Fig. 4.18, since the most costly part of the algorithm for anisotropic etchants is avoided with isotropic solutions, the task of updating the SFM lists (i.e. steps 17 to 19 of algorithm 15) is now the most computationally expensive part requiring 39.2% and 35.8% of the total simulation time respectively for experiments (a) and (b) of Fig. 4.17. Furthermore, the calculation of spatial derivatives is also relevant, taking 33.8% and 33.5% of the simulation times.

4.5.4 GPU CCA vs GPU SFM

In this section, the GPU implementation of the SFM presented in section 4.4.2 and the compensated CTS-CCA [292] are compared in realistic scenarios, including direct comparison with experiments. The goal is to validate the developed GPU implementation by simulating wet etching experiments to fabricate MEMS structures in silicon and quartz substrates using several etching solutions. Hence, the results of both approaches are compared in terms of accuracy of the results, i.e. by comparing measurements with experimental ones, and in terms of simulation time since both simulators are implemented in CUDA C. Notice that the CCA implementation makes use of an octree data structure, which reduces the memory usage but adds a relevant overhead of the GPU calculations [102].

In order to properly compare the accuracy and the execution time of both approaches, the grid sizes of both methods are determined in the context of Fig. 4.15(b), i.e. to obtain a 10% error when measuring accurate features. These grid sizes ensure accurate results even in those sensitive parts of the structures.

The testing machine of the benchmark experiments consists of an Intel Core i7 at 2.8 GHz with 4 GB of RAM using 64 bit Windows-based and an Nvidia GeForce GTX 260 GPU device.

Fig. 4.19 collects all the simulated experiments. From left to right, the columns correspond to (i) the required mask patterns, (ii) the experimental results, (iii) the SFM-simulated fronts and (iv) the CCA-based results. According to this figure, the structures obtained by SFM and CCA are very similar. This is confirmed by the measurements shown in the corresponding pictures, where the differences between both simulators are of the order of a few to several tens of microns, while the substrates measure several hundreds to even a few thousand microns. The feature dimensions shown in red on the experimental images have been obtained by using the scale attached to each figure. For completeness, table 4.5 collects information about the simulations, including the crystallographic orientation of substrate, etchant, substrate size, etching time (for every etching process, if more than one are required), the grid size used in both simulation models and the computational time required by both methods.

In contrast to complex MEMS structures, such as experiments (a), (b), (c), (g) and (h), a much smaller grid could be used without sacrificing accuracy in the case of simpler topologies with no sensitive details, such as experiments (d)-(g).

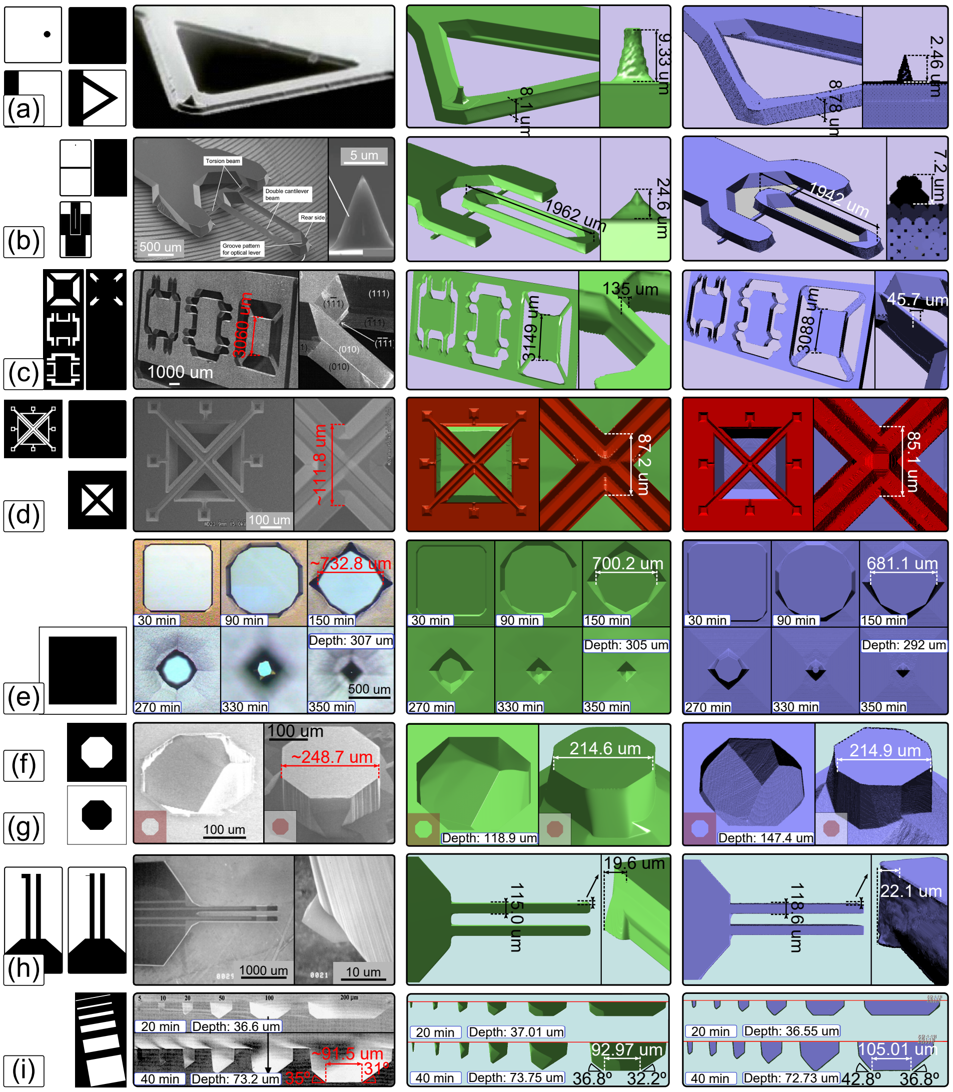


Figure 4.19: Comparison between experimental and simulated complex structures etched in different etchant solutions using silicon and quartz substrates: (left columns) applied masks, (center-left) experimental results, (center-right) SFM results, (right) CCA results. Experiments on silicon: (a) AFM tip [284], (b) dual-axis micromechanical probe [173], (c) three-axes accelerometer [142], (d) suspended microchannel [195] and (e) microneedles [174]. Experiments on quartz: (f) and (g) cavity and mesa [294], (h) tuning-fork probe [141] and (i) grooves [295]. Several length measurements are shown on the experimental and simulated scenes. Reproduced from [19].

	Experiment features				Surface grid size		Sim. time (s)	
	Material and orientation	Etchant	Substrate size (μm)	Etching time (min)	SFM (voxels)	CCA (UCs)	SFM	CCA
(a)	Silicon (100)	KOH 40 wt% 70 °C	250x250x22	$17 + 11 + 7$	536x536	375x375	8.1	70.2
(b)	Silicon (100)	KOH 40 wt% 70 °C	2266x4000x190	80 + 270	403x712	282x498	8.0	74.5
(c)	Silicon (100)	KOH 40 wt% 70 °C	21546x7182x399	114 + 256.5	928x309	648x216	2.1	21.7
(d)	Silicon (100)	TMAH 25 wt% 80 °C/ Triton 0.1 v/v add.	800x800	40 + 160	536x536	375x375	32.8	36.9
(e)	Silicon (100)	KOH 30 wt% 80 °C	1000x1000	30 to 350	536x536	375x375	63.2	157.9
(f)	Quartz (0001)	NH_4HF_2 85 °C	1200x1200	220	536x536	403x348	44.5	37.1
(g)	Quartz (0001)	NH_4HF_2 85 °C	1200x1200	220	536x536	403x348	52.2	125.5
(h)	Quartz (0001)	NH_4HF_2 85 °C	1353x2459x100	100	398x723	259x545	9.7	18.6
(i)	Quartz (0001)	NH_4HF_2 85 °C	723x390	20, 40	729x393	475x295	16.8	28.0

Table 4.5: Experimental conditions and simulation parameters used in Fig. 4.19.

Nevertheless, the same grid size criteria has been used for all the systems in order to be consistent.

The following is a more detailed description of the similarities and discrepancies between the experiments and the simulations. In addition, a description of the micromachining process of every experiment is commented.

AFM tip: 4.19(a)

This result is obtained by applying consecutive etching steps to a double-sided silicon wafer (100)-oriented. Consequently, each CCA UC is formed by 8 atoms. In particular, the following process has been utilized by both approaches:

- Define a double-sided (100) silicon wafer with 250x250x22 μm^3 dimensions.
- Apply the upper-left mask shown in Fig. 4.19(a) on top of the wafer.
- Apply the upper-right mask on bottom of the wafer.
- Wet etch with KOH 40 wt% 70 °C for 17 minutes.
- Remove top mask.
- Apply lower-right mask on the top side of the wafer.
- Wet etch with KOH 40 wt% 70 °C for additional 11 minutes.
- Remove the pattern shown in the lower-left mask from the bottom surface.
- Wet etch with KOH 40 wt% 70 °C for additional 7 minutes.
- Remove top and bottom masks.

Although no information about the etching steps or feature dimensions is provided by the source [284], the results demonstrate that a similar structure can be achieved by both simulators by appropriately selecting the parameters, i.e. the etchant, concentration, temperature, mask patterns and process times. The tip measurements shown in Fig. 4.19(a) differs 6.87 μm between both methods, producing the CCA a more realistic shape in this case. Nevertheless, the width substrate measurements are in close proximity, only differing 0.68 μm.

Dual-axis probe: 4.19(b)

Likewise the previous experiment, the substrate of this experiment is a double-sided (100) silicon wafer and each UC is formed by 8 silicon atoms. The micromachining process is [173]:

- Define a double-sided (100) silicon wafer with 2266x4000x190 μm^3 dimensions.
- Apply the upper-left mask shown in Fig. 4.19(b) on the bottom of the wafer.
- Apply the upper-right mask on the top of the wafer.
- Wet etch with KOH 40 wt% at 70 °C for 80 minutes.
- Remove top and bottom masks.
- Apply upper-right mask on the bottom side of the wafer.
- Apply lower mask on the top side of the wafer.
- Wet etch with KOH 40 wt% at 70 °C for additional 270 minutes.
- Remove top and bottom masks.

The two simulated shapes are very similar to the experimental structure. Due to the perspective of the experimental image it is difficult to determine the beam width. However, its length is about 2000 μm, in good agreement with both simulated values. The tip height obtained by the CCA simulator (7.2 μm) resembles the experiment better (9.1 μm) than the SFM (24.6 μm).

Three-axis accelerometer: 4.19(c)

For this experiment the same substrate than previous examples is used, i.e. a double-sided (100) silicon wafer. The fabrication process follows the next steps [142]:

- Define a double-sided (100) silicon wafer with 21546x7182x399 μm^3 dimensions.
- Apply the left mask shown in Fig. 4.19(c) on bottom and top sides of the wafer.
- Wet etch with KOH 40 wt% at 70 °C for 114 minutes.
- Remove the pattern shown in the mask on the right from top and bottom surfaces.
- Wet etch with KOH 40 wt% at 70 °C for additional 256.5 minutes.

- Remove top and bottom masks.

The experimental scale bar indicates that the width of the inertial mass obtained by CCA (3088 μm) is closer to the experiment. Nevertheless, the SFM value (3149 μm) differs by only 89 μm, while the length of the substrate is $\sim$ 22000 μm. Similarly, the width of the beam produced by the CCA approach is more accurate than the SFM one.

Suspended microchannel: 4.19(d)

Although in examples (a)-(c) the silicon surface is (100)-oriented and each UC of the CCA approach is formed by 8 atoms, in cases (d) and (e) the substrate surface is also (100) but each UC is formed by only 4 atoms. This is because the wafer is rotated 45° with respect to the previous ones, producing a different UC due to crystallographic structure of silicon [187].

The fabrication steps of this suspended microstructure are [195]:

- Define a simple (100) silicon surface with 800x800 μm^2 dimensions.
- Apply the upper-left mask shown in Fig. 4.19(d) on the top side of the wafer.
- Wet etch with TMAH 25 wt%+Triton 0.1 v/v at 80 °C for 40 minutes.
- Mask the top of the wafer with the upper-right pattern.
- Remove the lower pattern from the top side of the substrate.
- Wet etch with TMAH 25 wt% 80 °C for additional 160 minutes.

Notice that in this experiment, the applied mask forms part of the final structure. In particular the first wet etching process is used to form the channel itself. Then, the second etching enables the release of the microchannel by removing the material below the mask. Regarding the simulated results, both structures well reproduce the experimental one. The errors of the shown measurements are only 1.2 and 2.7 μm for SFM and CCA, respectively, whereas the length of the substrate is 800 μm.

Microneedle: 4.19(e)

This experiment consists in fabricating a microneedle with a simple wet etching step [174] by etching a silicon wafer masked with a squared pattern. The same substrate properties than previous example are used. This experiment takes advantage of the underetching properties of the KOH to produce the microneedle. In order to properly visualize this formation, the structures at several wet etching times are shown in Fig. 4.19(e). Accordingly, the micromachining process is:

- Define a simple (100) silicon surface with 1000x1000 μm^2 dimensions.
- Apply the mask shown in the figure on the top side of the substrate.
- Wet etch with KOH 30 wt% at 80 °C for the corresponding minutes according to Fig. 4.19(e) (from 30 to 350).
- Remove the top mask.

This experiment is well reproduced by both simulators. The displayed dimensions demonstrate that the SFM result is closer to the experiment, especially the reached depth, which only differs by 2 μm in comparison to the large substrate size. Nevertheless, the overall shape of the resulting structures are very similar with the experimental ones.

Quartz cavity and mesa: 4.19(f) and (g)

Since both experiments are essentially the same but with different mask patterns, they are commented together. The material employed as substrate for these experiments is (0001) quartz. Each UC used by the CCA approach for simulating quartz-based experiments contains 4 atoms. The simple fabrication process is [294]:

- Define a simple (0001) quartz substrate with 444x444 μm^2 dimensions.
- Apply on the surface the mask shown in Fig. 4.19(f) for the cavity or the one shown in (g) for the mesa experiment.
- Wet etch with saturated NH_4HF_2 at 85 °C for 81.5 minutes.
- Remove the applied mask.

The simulated shapes are very similar to the experiments. Although both simulators generate a similar width for the mesa structure, they differ by about 34 μm from the experimental mesa. These experiments validate the developed SFM applied to micromachining of real quartz-based structures.

Tuning-fork probe: 4.19(h)

The tuning-fork probe shown in Fig. 4.19(h) is fabricated on a double-sided (0001) quartz substrate. Despite the complexity of the experiment, this structure is directly fabricate with only one etching step. The micromachining procedure used by both simulators is [141]:

- Define a double-sided (0001) quartz substrate with 1353x2459x100 μm^3 dimensions.
- Apply on the top-side the left mask shown in Fig. 4.19(h).
- Apply on the bottom-side the right mask.
- Wet etch with saturated NH_4HF_2 at 85 °C for 100 minutes.
- Remove the applied masks.

The simulated structures are very similar to each other and also to the experiment as demonstrate the measurements. Due to the perspective of the experimental close-up (right-hand side image), it is difficult to obtain the tip length but it can be estimated that its length is lower than 10 μm. Although the SFM value is smaller than that for CCA (19.6 against 22.1 μm) and, thus, closer to the experiment, the overall CCA shape is more similar to the experiment.

Quartz grooves: 4.19(i)

Finally, another one-sided quartz substrate experiment is simulated. This experiment shows the different crystallographic planes appearing when etching a (0001) quartz surface [295]. The fabrication process is:

- Define a simple (0001) quartz substrate with 723x390 μm^2 dimensions.
- Apply the mask shown in Fig. 4.19(i) on the surface.
- Wet etch with saturated NH_4HF_2 at 85 °C for 20 and 40 minutes.
- Remove the applied mask.

The figures of the results show the cross section of the structure after 20 and 40 minutes of etching. As can be observed, the depths reached by both simulators are in close proximity to the experiment. Nevertheless, the largest cavity measurements show that the SFM method provides more accurate results than the CCA approach since both, the groove length and the formed angles, barely differ from the experimental ones.

It can be concluded that both the CCA and the SFM have reproduced successfully every tested experiment, proving the efficacy of the developed GPU SFM-based implementation. The maximum discrepancies occur at the smallest parts, such as the tips in experiments (a) and (b), where the two models differ by as much as 6.87 μm and 17.4 μm, respectively.

Furthermore, it must be considered that, probably, the conditions of the experiments were not exactly the same to those at which the etch rates were obtained, since every experiment has been performed by different researchers and laboratory equipment. Hence, this specific conditions cannot be taken into account by the simulators producing slightly different results. In addition, in some cases the masks were only provided with low quality or even not provided at all. As a consequence, some simulated measurements differs from experimental ones. This discrepancies can be reduced by performing own wet etching processes under strict conditions and properly selecting the parameters simulation. Hence, the most important conclusion is that Fig. 4.19 demonstrates the reliability of the proposed SFM simulator, which is capable of achieving similar accuracy while producing less noisy results than the CCA model. Although the CCA method provides better accuracy at small parts, the SFM implementation achieves better results for several systems, such as (e) and (i). Thus, we conclude that the proposed parallel GPU SFM implementation of the LS method has great potential to become a tool for the design of complex MEMS structures.

In terms of computational performance, our parallel SFM implementation is about $\sim 8.5 - 10$ times faster than the CCA method for those experiments where a CCA UC is formed by 8 atoms, i.e. examples (a), (b), and (c) of the Fig. 4.19. However, for the rest of the cases where a UC is formed by only 4 atoms, (i.e. experiments (d) to (i)) the SFM is only $\sim 1.1 - 2.5$ times faster and even 1.2 times slower for the example (f). Although the sequential implementation of both methods showed in section 4.5.3 that the CCA can be up to 2 times faster for anisotropic etchants, the GPU implementation of the CCA makes use of an octree data structure which adds an overhead to the GPU calculations, thus, increasing the execution time but reducing the memory usage [102].

Similarly to the sequential approach, the computational cost of the developed GPU SFM implementation is primarily due to the calculation of spatial derivatives, local etch rate and viscosity factors, which altogether are included in kernel 1 (alg. 23) and represent $27 - 49\%$ of the total computational time, as can be visualized in Fig. 4.20. According to the figure, steps 13 (kernel 2, algorithm 24), 17 (kernel 3, algorithm 25), 19 (kernel 5, algorithm 27) of algorithm 15 are also relevant, representing $6-14\%$, $7-11\%$ and $5-10\%$ of the computational time, respectively. Another relevant part of the algorithm is labelled as *Rest of alg.* in Fig. 4.20, and this part includes step 3 to 5 and also the search of the maximum viscosity factors α_q among the maximum values of each execution thread performed by the CPU. All these tasks take $15 - 24\%$ of the total simulation time, even 49.1% is taken by this part of the algorithm in example (c) since the total GPU time is very small, thus increasing the CPU contribution. Nevertheless, the transfer of the

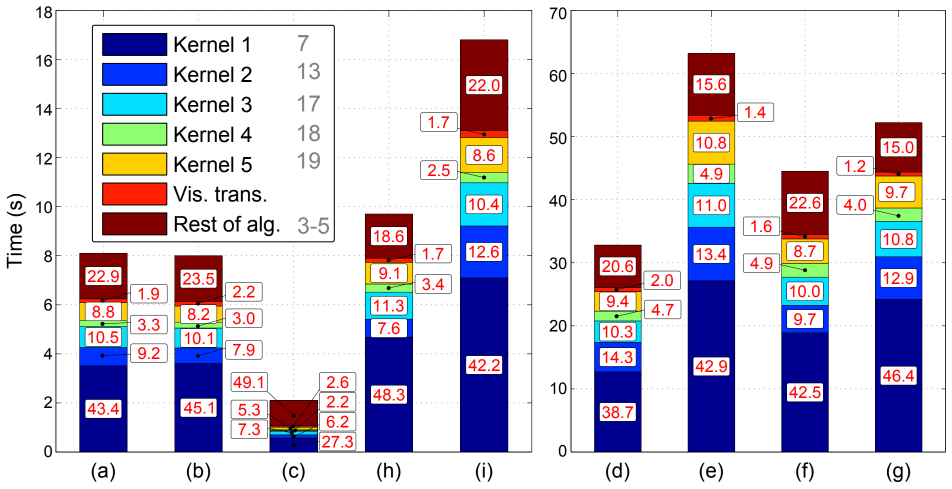


Figure 4.20: Representation of the execution times obtained with the GPU SFM implementation for the simulation results presented in Fig. 4.19. The red numbers indicate the contribution of the main parts of algorithm 15. The gray numbers indicate the corresponding step of the algorithm. The part labelled as *Vis. tran.* refers to the GPU to CPU transfers of the local maxima of the viscosity factors of every execution thread. The examples are grouped in two graphs with different time scales for a better visualization.

local thread maximum viscosity factors from GPU to CPU only takes $1.2 - 2.6\%$. Finally, step 18 (kernel 4, alg. 26) only represents $2 - 5\%$ of the global time.

4.6 Conclusions

In this chapter, wet etching process oriented to MEMS micromachining is simulated by means of the LS method. The aim is to provide a versatile tool for the design of MEMS devices based on the inherent capability of the LS method to simulate (i) the splitting and coalescing of disjoint regions of the front, such as in double-sided etching and (ii) new etchants and/or substrate materials without any need for recalibrating the internal parameters of the method. Based on the increased computational efficiency and accuracy with respect to the conventional LS method, an algorithm based on the SFM is presented. This SFM algorithm employs the etch rates directly obtained from experiments, contrary to the state-of-the-art atomistic methods like the CCA, which require a calibration process when the experimental conditions (e.g. temperature, etchant solution or substrate material) change. This algorithm is validated by the simulation of etching spherical samples with several different etchants and reconstructing the produced etch rate distributions. These distributions are compared with experimental and CCA results, proving that the proposed SFM produce very similar distributions than the experimental ones, and even less noisy results than those obtained with the CCA.

Accordingly, three implementations of the proposed SFM algorithm have been developed: (i) a purely-sequential Java-based, (ii) a parallel CPU version which takes advantage of the auto-vectorization of the code offered by some compilers and the implementation of a multi-threading based on the multi-core nature of modern CPUs, and (iii) a parallel GPU implementation where the most time-consuming tasks are efficiently computed by using the affordable, massively-parallel architecture of modern GPUs.

Since the three versions produce exactly the same results, their computational performance can be meaningfully compared. First, the two parallel implementations are compared to each other by simulating a set of simple etching processes. It is found that the GPU version results up to 7.4 times faster than the CPU implementation. Hence, it is concluded that the massively parallel platforms are more suitable for performing SFM simulations of wet etching.

The next performed comparison is between the sequential implementations of the proposed SFM algorithm and the state-of-the-art CCA method. For this comparison a set of simple etching processes is used, proving the SFM achieves similar accuracy as CCA while producing less fluctuations in the etch front and requiring roughly 4 times less memory, even if SFM needs about double the resolution than CCA. Although for highly anisotropic etchants SFM tends to soften the corners and edges, reducing slightly its accuracy, the differences between the simulated features by SFM and CCA are of the order of a few microns for substrates measuring even thousands of microns. In terms of computational performance, CCA is up to 2 times faster than SFM for anisotropic etchants while SFM becomes up to 10 times faster than CCA for isotropic etchants, for which SFM provides a smooth alternative to the noisy CCA results.

Finally, the GPU SFM implementation is compared with a GPU version of the CCA for a wide variety of experimental conditions, including silicon and quartz substrates in different etchants, such as KOH, KOH+IPA, TMAH and TMAH+Triton for silicon and NH_4HF_2 for quartz. Similarly to previous comparison, for highly anisotropic etchants SFM tends to soften the corners and edges, slightly reducing the accuracy. In addition, the SFM typically requires larger grids than CCA. The differences between the simulated features by both methods are of the order of several microns for substrates measuring even a few millimetres, concluding that the SFM implementation achieves similar accuracy as the CCA method with less fluctuation in the etch front. Due to the strong, parallel nature of the SFM and the high computational efficiency of the currently available, many-core platforms, such as Nvidia's GPUs, our parallel SFM implementation is typically faster than the CCA method. This feature is assigned to the use of an octree data structure in the CCA method, which reduces memory allocation but requires additional calculations and management.

In addition to the similar (or higher in some cases) computational performance, the greatest strengths compared to CCA of the proposed SFM implementations are: (i) the absence of a time-consuming calibration procedure prior to performing the simulations, which is strictly necessary in the CCA approach when the etchant is modified, (ii) the direct application of the simulation tool to any type of substrate, which typically requires a dedicated effort to analyse and classify the different atomistic neighbourhoods in the CCA approach, (iii) the smaller use of memory in comparison to CCA, and (iv) the faster simulation of isotropic etchants. For these reasons, the proposed SFM implementations provide accurate and fast simulations and they can result very valuable for MEMS design.

Chapter 5

Improvement of profile evolution in dry etching simulation by means of the Level Set method

The goal of this chapter is to improve current dry etching simulators that use explicit representation techniques to evolve the surface being etched. Explicit parametrization has some limitations, such as tracking the interaction of various fronts, since additional subroutines must be employed to handle such situations.

A LS implementation is proposed to use the current dry etching models while taking advantage of the implicit front representation strategy. In particular, an algorithm based on image reconstruction is developed to evolve the surface being etched according to Anetch models, a silicon dioxide plasma etching simulator. Based on the SFM, the proposed implementation is further optimized by implementing a parallel algorithm in order to execute many operations simultaneously on modern GPUs, reducing drastically the computational time. Hence, finally a robust three-dimensional and fully-operation dry etching simulator is obtained.

First, an introduction to plasma etching simulators is commented in section 5.1, including the problems of explicit surface representation as well as Anetch tool and previous LS-based simulators. Then, in section 5.2 the proposed algorithm for surface evolution is explained, including the parallel GPU strategy employed. Section 5.3 contains the details of the developed module for extracting the implicit updated surface and make it suitable for Anetch to apply again the etching models. Simulation results are collected in section 5.4 as well as a comparison with an experimental result. Finally, conclusion are commented in section 5.5.

5.1 Introduction

Plasma etching or dry etching processes are widely used in the fabrication processes of MEMS and semiconductor manufacturing industry for high fidelity pattern transfer and anisotropic etching. In contrast to wet etching process, plasma etching can usually obtain high anisotropy and high-aspect-ratio holes [296]. In micromachining processes, it is very useful to predict the resulting structure of an experiment under specific conditions so higher accuracy can be achieved and costs can be reduced. Correspondingly, several models of plasma etching have been developed in the last years [247, 258, 297–299].

Particularly, this chapter is focused on the RIE process simulation. In this etching process a chemically reactive plasma is used for removing material, usually deposited on wafers. Both, physical and chemical reactions contribute to the etching process, enabling the creation of high anisotropic etchings. More details about equipment and involved reactions are explained in section 2.4.4.

As commented in section 2.4.4.1, a RIE simulation tool can be understood as a whole process formed by several modules. First, physical parameters such as incident particle fluxes, yields, angular distribution and energy, are obtained directly from experimental measurements or from equipment simulation. This generic data is then utilized by the next module, which considers the current surface topography and calculates the local fluxes of the different species. Then, the corresponding local etch rates are calculated by the next module. Finally, the profile evolution module evolves the surface according to the new calculated etch rates.

According to section 2.4.4.1, there are mainly two approaches to represent a surface in order to evolve it: explicit parametrization and implicit representation. When evolving a surface explicitly parametrized, motion is applied to the actual points that form the surface. This approach requires additional programming effort to handle topological changes in the surface, such as coalescing or splitting of contiguous regions, otherwise, unrealistic results can be produced. On the other hand, implicit surface representation approaches like the LS method, embed the surface inside a higher-dimensional function, such that the motion is applied to this function. This representation enables a natural description of topological changes without any further programming and computational effort. Additionally, the parallel nature of the LS method enables further optimizations such as the execution in a parallel environment like GPUs.

This chapter is focused on the profile evolution stage of RIE simulators by means of the LS method, enabling the simulation of complex processes. Some LS-based RIE simulators have been presented. First, the LS method was proved to be able to emulate a very simple directional etching [16]. Later, Hwang et al. used this method for simulating plasma etching of silicon [300]. Then, Im and Hahn presented a two-dimensional silicon dry etching simulator which considers several plasma parameters under various conditions [301]. After that, Kokkoris et al. published a modular tool based on the LS method capable of reproducing experimental phenomena such as the RIE lag and inverse RIE lag effects [257]. Nevertheless, no optimization was implemented for the LS method. Accordingly, Radjenović et al. implemented a SFM-based tool capable of emulating high anisotropic etching processes by using a simple model as the LS velocity function such that the results resembled plasma etching processes [258]. This implementation was improved for silicon dioxide etching by including a Monte Carlo module that considers different species [302]. Finally, Ertl et al. presented a SFM-based RIE simulator of silicon coupled with a Monte Carlo module for flux calculation [260, 303, 304].

Although all of this LS-based implementations are capable of obtaining results that resemble RIE process results, they have never been directly compared with experimental results and no computational performance analysis have been presented. Accordingly, in this thesis a parallel GPU SFM implementation based on the surface reconstruction technique of chapter 3 is proposed to implicitly evolve the surface being etched according to etch rates provided by previous modules. In particular, this study is focused on a silicon dioxide etching in fluorocarbon plasma process simulator, which nowadays is included in the software Anetch [261]. Currently, Anetch uses only explicit surface representation, producing unrealistic results for some etching processes simulations, as shown in Fig. 5.1.

The whole simulating tool proposed consists of three modules: the Anetch models, the SFM evolution surface tool and the extraction of implicit surface. Anetch takes equipment parameters and the current structure, defined by vertices and faces, and calculates the new position of every vertex. These new coordinates are introduced to the SFM module that evolves the surface up to the new vertices positions, thus, ensuring that the SFM-updated surface is a realistic surface, contrary to explicitly parametrized surfaces which can produce results without physical sense (see Fig. 5.1). Finally, the implicit surface is extracted from the LS function and new faces and vertices are generated. Then, the new structure can be used as input for the Anetch models and repeat the process until the desired depth is reached. This algorithm is shown in Fig. 5.2. This enables the possibility of using etching models that need an explicit representation of the structure, like Anetch, in combination with the SFM module, which evolves the surface according to the etching model.

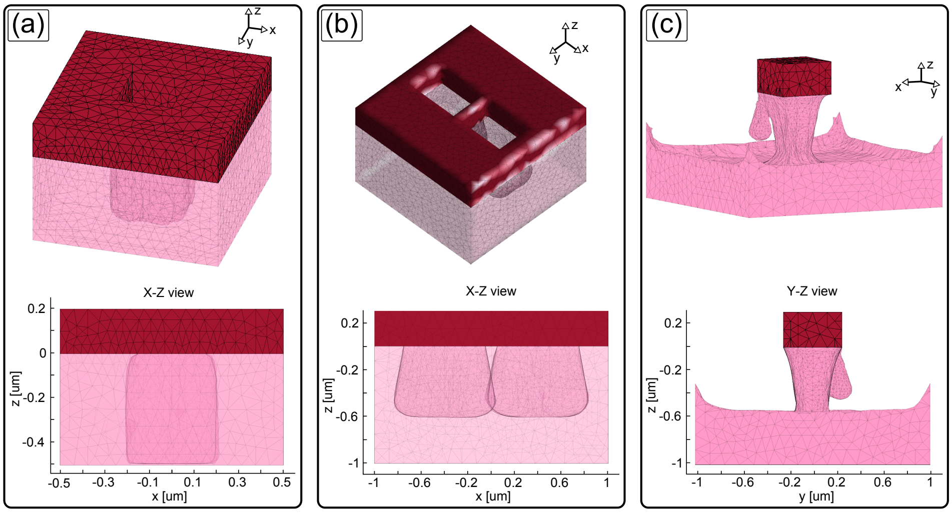


Figure 5.1: Anetch unrealistic simulation results. Because of surface explicit parametrization, (a) a through hole cannot be simulated since there is always remaining material at the bottom of the hole, (b) the interaction of two surfaces is not properly emulated, and (c) an unrealistic artefact is created during the simulation.

5.1.1 Anetch

The Anetch models considers neutral and ionic species. These species are formed during plasma bulk reactions and they are fractions of the original precursor molecules. In this study only C_2F_6 is used as precursor. Concentration information and angular distribution of neutral and ionic species can be obtained from equipment simulation or experimental measurements.

A polymer layer between oxide and plasma is formed during the oxide etching process. This polymer is involved in some reactions and it reduces reaction rates due to dissipated energy (for ions) or diffusive loss (of all species before they reach the polymer-oxide interface where they can react) having an effect on the etching process. The processes are modelled by the equations described by Zhang and Kushner [305]. These considered reactions are shown in Fig. 5.3.

The input parameters taken by Anetch are the current structure explicitly parametrized and the relative fluxes of the different species (neutrals and ions), i.e. F, CF, CF_2, C_2F_3, C_2F_4, CF_2^+, CF_3^+, $C_2F_4^+$, and $C_2F_5^+$. This information is provided by equipment simulation or experimental measurements, and it is the field values of ions and neutrals fluxes to the surface. For neutrals species an isotropic angular distribution is assumed, while for ions a Gaussian curve is used. According to this input, the model makes an initial guess of the local fluxes of

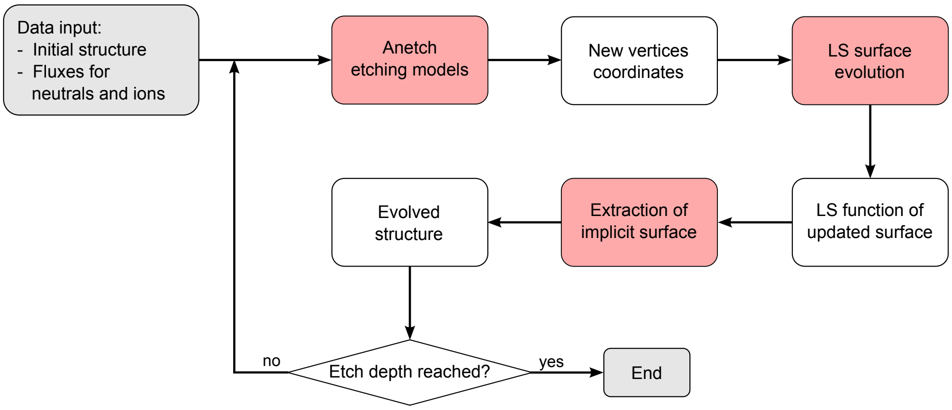


Figure 5.2: Workflow diagram of the general simulator. The red boxes are the three modules of the simulator, whereas white boxes represent the corresponding result of each module.

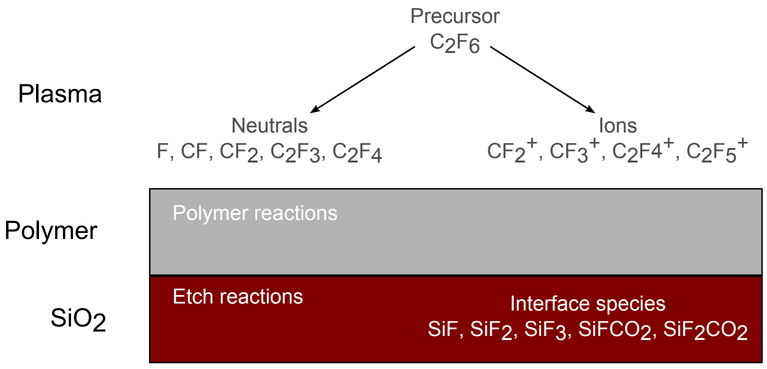

Precursor
C_2F_6
Plasma
Neutrals
F, CF, CF_2, C_2F_3, C_2F_4
Ions
CF_2^+, CF_3^+, $C_2F_4^+$, $C_2F_5^+$
Polymer
Polymer reactions
SiO_2
Etch reactions
Interface species
SiF, SiF_2, SiF_3, $SiFCO_2$, SiF_2CO_2

Figure 5.3: Underlying mechanism as implemented by Anetch for the oxide etching simulation.

all different species (ionic and neutrals) over the whole initial three-dimensional structure. Whereas the initial guess values of the fluxes are obtained directly from the user input (experiment or equipment simulation as stated above), the concentrations of the species at the polymer-oxide interface and the polymer layer thickness need to be derived for the initial guess. The basic principle for the initial guess of these derived quantities is to set up a system of equations according to the interaction between the different species assuming steady state for all concentration and flux values [305]. This means that for all species, consumption must be equal to production, such as for the polymer reactions [305]. For the calculation of this initial guess, fluxes from different surface locations are not considered, which will be done in the following steps for taking into account the three-dimensional topology of the structure.

Based on all values of the initial guess, local sticking coefficients for all species are calculated. In the first step they will not depend on the surface position, since all surface positions of the three-dimensional feature have the same initial guess values. As an example, the equation that relates the local sticking coefficient for the F species (fluorine radicals) to polymer thickness (number of polymer monolayers $[P]$), and surface concentrations is given:

$$\begin{aligned} SC_F = \; & [P]K_{10} \\ & +\theta_{SiF}K_{11} + \theta_{SiF_2}K_{12} + \theta_{SiF_3}K_{12} \\ & +\theta_{SiF_2CO_2}K_{12} + \theta_{SiFCO_2}K_{12} \end{aligned} \tag{5.1}$$

where SC_F is the sticking coefficient of F, θ represents the surface coverage (i.e. concentrations) of the corresponding species and K_i are proportionality constants of the model that depends on the energy of the ions (if ions are involved) and the thickness of the polymer layer (for reactions at the polymer-oxide interface) [305]. Similar equations are used for describing the rest of species.

The next step is to calculate a new local fluxes solution of all the considered species according to sticking coefficients. This means that adsorption and desorption of the species are considered, leading to varying flux values inside of the three-dimensional structure. This is the most computationally expensive part of the Anetch algorithm since, for every face of the surface being etched, the solid angle is treated in discrete portions to each of which a certain value for the flux arriving from another surface face at this spatial direction is attributed. Then, these local fluxes are used as the initial guess for the next iteration.

In the next iterations, the sticking coefficients are again calculated but, now, based on the updated values for fluxes, concentrations and polymer thickness. Then, with the new values for the sticking coefficients a new solution for the local fluxes is determined, and so on. The criterion for termination is the change between the relative values of the fluxes (i.e. the flux value is set to unity in case the flux is equal to the input value mentioned above) of two subsequent iterations. If the

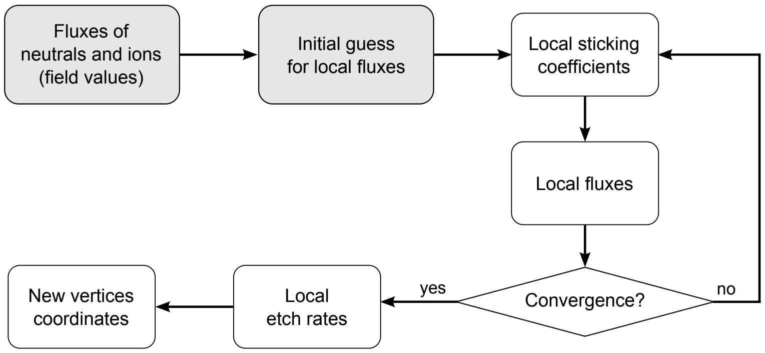

Fluxes of neutrals and ions (field values)
Initial guess for local fluxes
Local sticking coefficients
Local fluxes
Convergence?
no
yes
Local etch rates
New vertices coordinates

Figure 5.4: Workflow diagram of Anetch silicon dioxide simulator. This corresponds to the first module of the whole program.

maximum (taken over the different species) of this difference is below a threshold, convergence is defined to be reached. Then, corresponding local etch rates values are determined, which allows to calculate local shifting distances for an etching time step [305].

These new vertices are taken by the LS module to evolve the surface and then, the new structure is extracted. This is repeated until the desired depth is reached. Fig. 5.4 shows the steps of this first module, i.e. the Anetch models.

5.2 Surface evolution module

Instead of using explicit parametrization techniques to evolve the surface being etched, an implementation based on the surface reconstruction algorithm explained in chapter 3 is proposed. The essential idea is, considering the surface of the current structure, Anetch produces the new unconnected points of the surface according to the etching models, then, the current surface is embedded inside a SDF and evolved up to the new points with the LS reconstruction algorithm.

Particularly in the proposed surface reconstruction process, the coordinates of the vertices produced by Anetch (from now on referred as set P) are used to indicate the final position of the updated surface. Therefore, given an initial surface Γ_0 formed by connected points, Anetch module calculates the new coordinates that must be used to form the updated surface, but these points P are unconnected. Then, Γ_0 is embedded inside the ϕ function and, it is updated by means of the LS method until the position indicated by P is reached by the zero-level (i.e. the embedded surface). The evolved surface embedded inside ϕ is then extracted, producing Γ_1. In Fig. 5.5 a simple two-dimensional example of this procedure

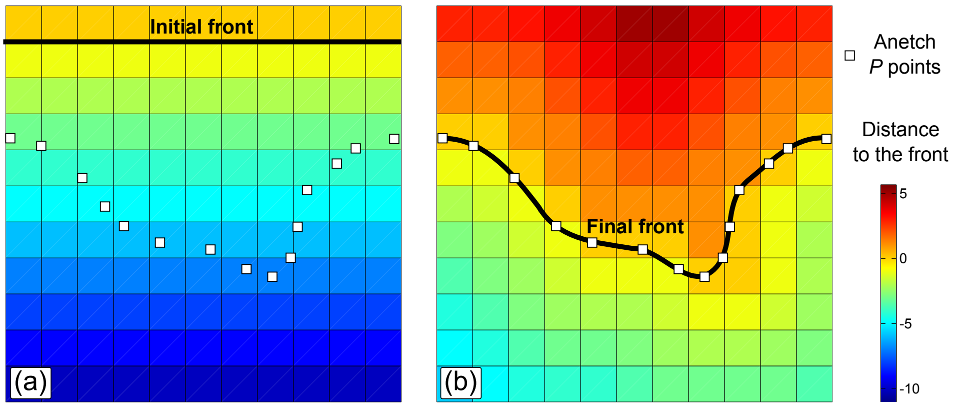


Figure 5.5: Two-dimensional representation of a simple surface evolution example with the LS method: (a) initial front guess, (b) final state of the front adapted to the P points. The voxel colors represent the value of ϕ. Anetch takes the initial front and produces the unconnected P points, then, the this front is evolved by means of the LS method up to the P points.

is represented. In the next iteration, Anetch will use Γ_1 to calculate the new coordinates that will be used to indicate the new position of Γ_1, which is already embedded in ϕ from the previous iteration. Thus, ϕ is updated again to produce Γ_2. This process is repeated until the desired depth is reached, as depicted in Fig. 5.2.

In the implementation developed in chapter 3, the surface to evolve is a closed surface surrounding the unconnected points (see Fig. 3.2). Nevertheless, in this chapter the P points are located under the surface, which is not closed but limited by the grid boundaries as shown in Fig. 5.5. In this chapter, the convection model explained in section 3.2.2 in combination with the SFM is utilized since it has been proved to be sufficiently accurate while requiring less computational effort. Thus, the LS function is updated according to:

$$\phi^{n+1} = \phi^n + \Delta t[\nabla d(\vec{x}) \cdot \nabla \phi], \tag{5.2}$$

where $d(\vec{x})$ is the matrix distance between every grid point $\vec{x}$ and the closest P point. For determining the derivatives of the term $\nabla d(\vec{x}) = (D_x, D_y, D_z)$, second-central order differences (2.28) are used. In addition, forward first-order Euler scheme is employed for time discretization, in combination with the upwind differencing scheme (see algorithm 1) to properly select the forward (2.26) or backward (2.27) first-order derivatives of the term $\nabla\phi$. As a consequence of using the convection model, the time step is chosen according to the next CFL condition:

$$\Delta t = \frac{0.5\Delta x}{\max\left\{|D_x| + |D_y| + |D_z|\right\}}, \tag{5.3}$$

Vertices(Nv){
x_0, y_0, z_0
x_1, y_1, z_1
...
x_{Nv}, y_{Nv}, z_{Nv}
}
Edges(Ne){
v_0, v_1
v_1, v_2
...
v_{Nv-1}, v_0
}

Faces(Nf){
e_0, e_1, e_2
e_2, e_3, e_4
...
e_{Ne-1}, e_0, e_1
}
Elements(2){
$f_0, f_1,...$
$f_1, f_{Nf-1},...$
}
Material e_1 = Photoresist
Material e_2 = SiO_2

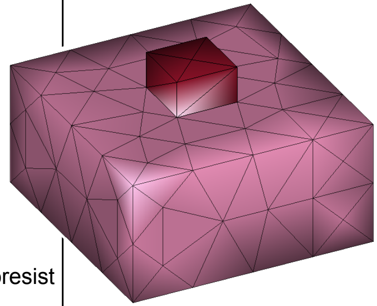


Figure 5.6: Simple structure formed by two elements: photoresist (dark red) and silicon dioxide (pink). The corresponding simplified DF-ISE file is presented. The variables in parentheses indicate the number of the corresponding item (vertices, edges, etc.).

ensuring convergence and numerical stability. Notice a uniform LS mesh is used such that $\Delta x = \Delta y = \Delta z$.

5.2.1 Input data

The LS surface evolution module takes two input files: the current structure explicitly represented and a list of points that must be updated, including the new positions of such points obtained by Anetch (P points). Anetch requires an explicit representation of the structure being etched, thus, the hierarchical format *DF-ISE* is utilized for this. This is the native file format of the ISE software company, which was acquired by the company Synopsys [306]. This data format contains information of every material that forms the structure, e.g. silicon dioxide and photoresist (mask). The basic information that requires a structure to be described with the DF-ISE format is: the coordinates of all vertices, followed by a list of edges, which are formed by connecting two vertices. Then, a list of all faces (formed by three edges) is included. Following, the different elements of the structure are defined by listing those faces that form each element. Notice that a face can be included in two different elements when it is at the interface between both elements. Finally, a material is assigned to each element of the structure. A simple structure formed by two different elements and the corresponding simplified DF-ISE file is presented in Fig. 5.6.

This explicit data must be represented in its implicit form such that the LS can properly update the surface. To accomplish this, the LS module first reads all the data stored in the DF-ISE file. In order to optimize computational performance and memory space, only those surface points that must be updated are computed by this module, i.e. those points of the top surface of the element being etched

(SiO_2) that are not covered by the masking material (avoiding to compute those points of the side walls and those of the bottom of the substrate since they must not be modified). Accordingly, x and y surface dimensions are obtained among such points. Additionally, the maximum and minimum distances between connected surface points ($\max_{dist}$ and $\min_{dist}$) are found. These values will be used for determining the mesh resolution and the initial LS surface. Likewise, the vertical z dimension of the LS grid is determined by the minimal z value of those P points included in Anetch's output.

With the maximum and minimum values for each spatial dimension, the boundaries of the LS grid are defined. On the other hand, grid resolution is chosen to be proportional to the minimal distance between connected points of the region being etched:

$$\Delta x = n \cdot \min_{dist} \tag{5.4}$$

The parameter n is chosen for each simulation depending on the desired accuracy. Therefore, the number of points in each spatial dimension is determined by:

$$\begin{aligned} num_x &= \left\lfloor \frac{max_x - min_x}{\Delta x} + 1 \right\rfloor \\ num_y &= \left\lfloor \frac{max_y - min_y}{\Delta x} + 1 \right\rfloor \\ num_z &= \left\lfloor \frac{max_z - min_z}{\Delta x} + 1 \right\rfloor \end{aligned} \tag{5.5}$$

where max_q and min_q are the maximum and minimum values of spatial dimensions $q = x, y, z$. Notice for the z values only the P points are considered, nevertheless, the whole surface substrate is covered by the LS grid to allow simulations of high underetching processes.

5.2.2 Distance matrix calculation

According to the convection model (5.2), the gradient of the distance matrix $d(\vec{x})$ is employed to update the surface. This matrix is the unsigned distance between every grid point and its closest P point. To solve this, the propagating algorithm 9 is utilized such that the S cloud of points is equivalent to the Anetch P points. The main idea of this algorithm is to discretize the P points over the grid to find those grid points that better approximate the P points. Then, the exact distance between the corresponding grid points and the closest P points is calculated. After that, these grid points propagate the coordinates of their closest P points to their neighbouring grid points so they can calculate the exact distance to the P points. This process is repeated until every grid point has a distance value assigned.

Then, the second-order central differences in each dimension are calculated, producing the three matrices D_x, D_y, and D_z.

5.2.3 Initial surface determination

In the first iteration of the whole simulation process (see Fig. 5.2), an initial surface must be determined. This initial surface is evolved up to the P points with (5.2). Similarly to the surface reconstruction algorithm, it is found that a proper initial surface is formed by those grid points that have a distance value $\epsilon = \max_{dist}/1.7$. For this task the tagging algorithm 10 is employed. This simple algorithm starts tagging all the grid points as *interior* but the points of the plane at the top of the grid which are tagged as *exterior* and added to a linked list. Then, for every point of the linked list, every neighbouring point is inspected: if it is tagged as *interior* and has a distance value higher than $\max_{dist}/1.7$, it is tagged as *exterior* and included in the linked list. Otherwise (i.e. it is *interior* with a distance value lower than $\max_{dist}/1.7$), it will be used as initial surface. After every neighbouring point has been visited, the linked list point is removed from it. This process is repeated until the linked list is empty.

Once the initial surface points are found, it has to be embedded inside a SDF and the grid points must be included in the corresponding SFM lists and labelled with the proper *state* according to algorithm 12 and taking into account the tags produced with the tagging algorithm.

This process only has to be performed in the first iteration. For the rest of the simulation process, the LS module takes the ϕ function resulting from the previous iteration as initial surface. Nevertheless, the LS grid is expanded in the vertical z dimension according to the new P points.

5.2.4 Evolution loop

After obtaining the initial surface and the SFM lists, the surface is updated according to the convection model (5.2) until convergence is reached. According to section 3.2.1, the energy of a surface is proportional to the sum of the distance values $d(\vec{x})$ of those points that form the surface. Using the convection model to evolve a surface, its energy is decreasing until a local minimum is reached, which corresponds with the surface attached to P points. Hence, in order to obtain such surface adapted to the points, after each SFM iteration, the surface energy is calculated and when the difference between the last iteration and the current one is small enough, the evolution loop is stopped.

An example of the surface energy evolution according to the convection model can be visualized in Fig. 5.7. The surface energy is calculated by summing the distance values $d(\vec{x})$ of all the surface points. Likewise, the surface points are those included in the SFM list L_0. The first iteration of the whole etching process (see workflow of Fig. 5.2) of a generic structure is shown in the left graph. Notice that all the tested structures have similar energy behaviour, thus, only one example

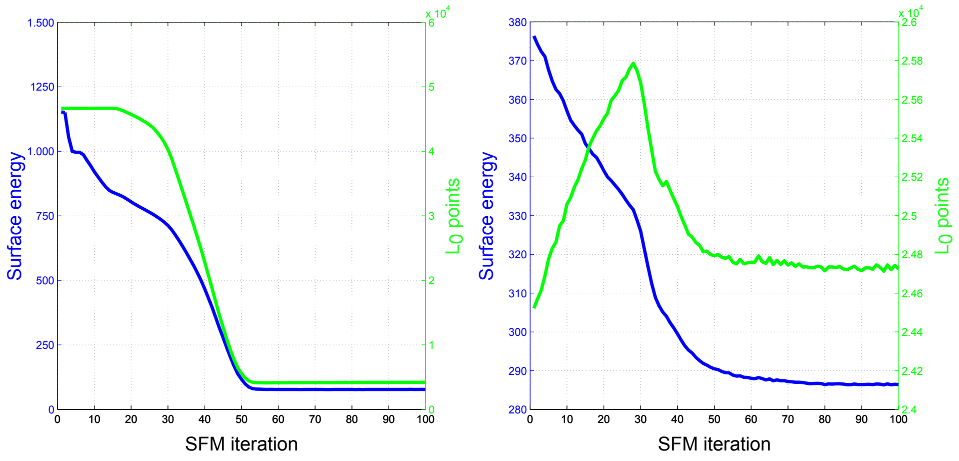


Figure 5.7: Evolution of the surface energy (blue) and the number of SFM L_0 list points (green) as the surface is evolved with (5.2). The left graph corresponds to the first iteration of the whole etching process whereas the right-side graph is the eleventh iteration.

is shown. On the other hand, the eleventh iteration is presented in the graph on the right. In both cases, the surface energy decreases as the surface is evolved and, usually, after 50 or 70 iterations approximately, the energy and the number of active points has converged. At this point, the local minimum energy which corresponds with the attached surface has been reached and the evolution surface process can be stopped. Notice at the beginning of the eleventh iteration, the number of points included in L_0 list is increased, this is because the etched surface has been expanded by Anetch and it covers more grid points.

5.2.5 Complete algorithm

Finally, the whole procedure of the profile evolution module by means of the SFM is presented in algorithm 28. The LS grid is generated according to the vertices of region to be etched, nevertheless, if it is not the first iteration, the x and y dimensions are directly obtained from the ϕ function of the previous iteration, whereas the vertical dimension z is updated according to P points. Similarly, if it is not the first iteration, the tasks of finding an initial surface and building the SDF are avoided since previous ϕ function is taken.

It is important to notice that Δt represents a step of an artificial time used by the LS method to evolve the surface in the evolution module. This artificial time step needs to be used in order to ensure numerical stability and convergence, however, the actual etching time is controlled by Anetch.

Algorithm 28: SFM implementation for profile evolution module.

Data: DF-ISE file describing the current structure

Data: Points to be updated and their new positions (P)

```
1  Read DF-ISE file and determine among the vertices of the region to be etched: the
   maximum and minimum values in $x$ and $y$ dimensions, $\max_{dist}$ and $\min_{dist}$.
2  Find the minimum $z$ value among the $P$ points.
   if first iteration then
3      Generate the LS mesh according to this maxima and minima, by using (5.4).
   else
4      Read the $\phi$ function from previous iteration and initialize SFM lists according to the
       value of every point.
5      Extend $z$ grid dimension according to the minimum $z$ value.
6  Build distance matrix $d(\vec{x})$ with propagating algorithm 9.
7  Calculate $\nabla d(\vec{x})$ with second-order central derivatives (2.28).
   if first iteration then
8      Find a proper initial surface with algorithm 10.
9      Build the SDF $\phi$ of the initial surface and add the points to the corresponding SFM
       lists with algorithm 12.
10 Calculate the time step of the convection model $\Delta t$ according to (5.3).
11 while no convergence do
       for each $L_0$ point $\vec{x}_i$ do
12         Calculate $\phi_x^+, \phi_x^-, \phi_y^+, \phi_y^-, \phi_z^+, \phi_z^-$.
13         Use $D_q(\vec{x}_i)$ to determine, by upwind differencing algorithm 1, the proper
           derivative to use in each dimension $q$.
14         Update implicit function $\phi$ with (5.2).
           if $\phi(\vec{x}_i) < -0.5\Delta x$ then
15             remove $(\vec{x}_i)$ from $L_0$ and add it to $S_{-1}$.
           if $\phi(\vec{x}_i) > 0.5\Delta x$ then
16             remove $(\vec{x}_i)$ from $L_0$ and add it to $S_{+1}$.
17     Update $L_{+1}$ and $L_{-1}$ lists with procedure 5.
18     Transfer points from auxiliary lists by applying procedure 6.
19     Add corresponding points to $L_{\pm 1}$ according to procedure 7.
20     Calculate surface energy $E^{n+1}$.
21     if $|E^{n+1} - E^n| < tol$ then break;
22 Go to extraction module.
```

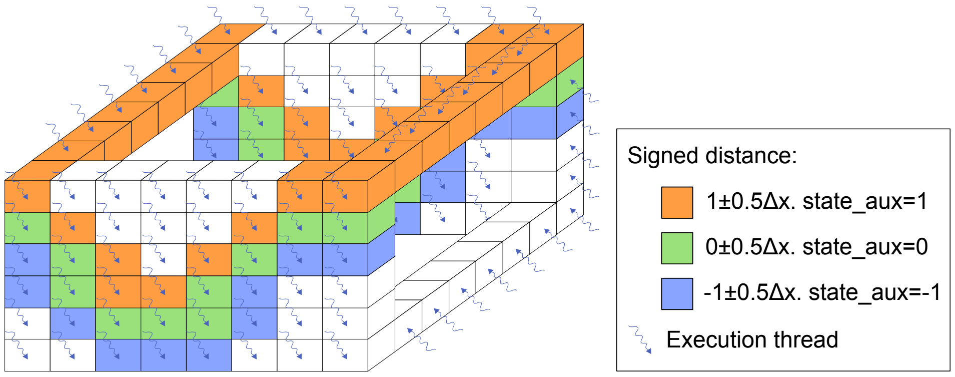


Figure 5.8: Simple three-dimensional example showing the execution thread distribution used for the SFM evolution profile module. Each execution thread is in charge of computing one grid point. Some intermediate voxels have been removed for a better visualization.

5.2.6 GPU implementation

In addition to the SFM optimization, a parallel implementation of the profile evolution module has been developed for a further reduction of the computational time. Based on algorithm 28, this implementations is written in CUDA C programming language and is executed on an Nvidia GeForce GTX Titan [307], which has 2688 CUDA cores and forms part of the Kepler microarchitecture (see section 2.3.3.3). Some features of these platform are collected in table 2.2.

The strategy used for wet etching simulation to emulate the SFM method on a GPU (see section 4.4.2) is used in this module. Thus, the three-dimensional auxiliary variable *aux_state* is employed to indicate the current SFM list of every grid point, thus denoting which points must be updated in each algorithm step.

Kepler microarchitecture devices enable the possibility of creating three-dimensional blocks of execution threads as well as three-dimensional grids of blocks. Furthermore, due to the high amount of threads that can be handled by such devices, one execution thread can be in charge of only evaluating one grid point, contrary to the implementation of section 4.4.2 where each thread has one grid column assigned. Therefore, there is no need to use upper and lower boundaries for grid columns since every thread visits a specific grid point and, depending on its state, will perform the corresponding operations. A three-dimensional simple example of the thread distribution can be visualized in Fig. 5.8.

Furthermore, in order to obtain the best performance, the thread block size must be multiple of the warp size (which is 32 threads). Since the number of maximum threads per block is 1024 for the Kepler microarchitecture, it has been found that 512 threads per block is the best choice. Thus, each block is formed by 8x8x8 threads. Therefore, the number of blocks in each dimension required to cover the whole LS grid is determined by

$$\begin{aligned} Nblock_x &= \left\lfloor \frac{num_x + 7}{8} \right\rfloor \\ Nblock_y &= \left\lfloor \frac{num_y + 7}{8} \right\rfloor \\ Nblock_z &= \left\lfloor \frac{num_z + 7}{8} \right\rfloor \end{aligned} \tag{5.6}$$

Thus, a three-dimensional grid of blocks is created, containing each block 512 (8x8x8) execution threads. This strategy ensures the creation of one execution thread per grid voxel.

5.2.6.1 Main variables

Although the requirements for obtaining a coalesced access to global memory has been reduced in Kepler microarchitecture [92], it is still important to pay attention to the memory access pattern in order to optimize memory bandwidth.

Similarly to the wet etching GPU implementation, the ϕ variable is stored in texture memory to explode spatial proximity of the threads accessing to such values, therefore, due to the thread distribution utilized and the requirements of the Kepler microarchitecture [92], coalesced accesses are obtained when accessing to data stored in global memory.

Table 5.1 collects all the main variables employed in the GPU implementation of the profile evolution module. Due to the upwind differencing technique used for updating the surface properly, the values D_x, D_y, and D_z are stored during the whole simulation in the GPU. Moreover, the variable D_{max} is used for obtaining the time step on the GPU, therefore minimizing the number of transfers between CPU and GPU device.

5.2.6.2 Workflow

To ensure a proper thread synchronization, algorithm 28 is divided in several CUDA kernels invoked by CPU. In the present implementation, not only the evolution loop process is executed on the GPU but the operations of calculating the term $\nabla d(\vec{x})$ (i.e. step 7 of algorithm 28) and the procedure of building the SDF of the initial surface and assigning the corresponding *state_aux* tags to every grid point in the first iteration (i.e. step 9) are performed in the GPU

Variable	Type	Size	GPU memory
ϕ	float	grid	Texture
ϕ_x	float	grid	Global
ϕ_y	float	grid	Global
ϕ_z	float	grid	Global
$state$	char	grid	Global
$state\_aux$	char	grid	Global
D	float	grid	Global
D_x	float	grid	Global
D_y	float	grid	Global
D_z	float	grid	Global
D_{max}	float	grid	Global

Table 5.1: Main variables allocated in GPU device according to the SFM parallel GPU implementation for RIE profile evolution. All the variables have *grid* size, i.e. they are three-dimensional matrices of $num_x \cdot num_y \cdot num_z$ entries.

device. Additionally, obtaining Δt (step 10) is also performed on the GPU, thus minimizing the number of data transfers between CPU and GPU. As a consequence, three additional kernels in comparison to wet etching implementation are developed for such operations. They are explained in algorithms 29, 30, and 31 respectively, and all the presented operations are performed by every execution thread. When launching a CUDA kernel, every created execution thread is assigned with an identifier (tb_{id}) that indicates the thread number within a block. In addition, a block identifier (b_{id}) is also assigned, thus, using both identifiers, an absolute unique thread identifier (t_{id}) is obtained, which can be used for accessing to global variables. In case that more threads than grid points would be created, the excess threads would not be computed, ensuring that every thread only accesses to the corresponding position.

Algorithm 29: Kernel 1 of GPU SFM implementation of profile evolution module. It corresponds to step 7 of algorithm 28.

```
Data: $t_{id}$ = Unique thread identifier
1 Calculate $D_x(t_{id}), D_y(t_{id})$, and $D_z(t_{id})$ with second-order central differences (2.28).
2 $D_{max}(t_{id}) = |D_x(t_{id})| + |D_y(t_{id})| + |D_z(t_{id})|$.
```

Regarding the kernel 2 (algorithm 30) used in the first iteration of a simulation for building the initial SDF and the *state_aux* matrix, notice the *state* tags produced by algorithm 10 are used.

Algorithm 30: Kernel 2 of GPU SFM implementation of profile evolution module. Determination of the SDF ϕ in the first iteration of the GPU evolution profile module implementation. It corresponds to step 9 of algorithm 28.

```
Data: $state = \{exterior, interior, IS\}$
Data: $t_{id}$ = Unique thread identifier
switch $state(t_{id})$ do
    case $IS$
1           $\phi(t_{id}) = 0$
2           $state\_aux(t_{id}) = 0$
3           break;
    case $exterior$
            if $any\ adjacent\ point\ of\ (t_{id})\ has\ state = IS$ then
4               $\phi(t_{id}) = \Delta x$
5               $state\_aux(t_{id}) = 1$
            else
6               $\phi(t_{id}) = 1.5\Delta x$
7               $state\_aux(t_{id}) = 2$
8           break;
    case $interior$
            if $any\ adjacent\ point\ of\ (t_{id})\ has\ state = IS$ then
9               $\phi(t_{id}) = -\Delta x$
10              $state\_aux(t_{id}) = -1$
            else
11              $\phi(t_{id}) = -1.5\Delta x$
12              $state\_aux(t_{id}) = -2$
13          break;
14 $state(t_{id}) = state\_aux(t_{id})$
```

In kernel 3 (algorithm 31), a different strategy, of that used for the rest of the kernels (i.e. 8x8x8 threads form a block and the grid dimensions are calculated with (5.6)), is employed for execution thread distribution. This kernel calculates the time step according to (5.3), which requires a search among all the grid points to find the maximum value $\max\{|D_x| + |D_y| + |D_z|\}$. Notice that this value has been calculated previously in kernel 1 (algorithm 29) for every grid point and they are stored in variable D_{max}. Thus, the task of kernel 3 is to find the maximum value of the variable D_{max}. Accordingly, one-dimensional blocks of 512 threads as well as a one-dimensional grid of blocks are created. The number of blocks of this grid is obtained with:

$$Nblocks_{\Delta t} = \frac{num_x \cdot num_y \cdot num_z + 255}{256}. \tag{5.7}$$

Algorithm 31: Kernel 3 of GPU SFM implementation of profile evolution module. It corresponds to step 10 of algorithm 28.

```
Data: $tb_{id}$ = Thread identifier within a block
Data: $b_{id}$ = Unique block identifier
Data: $t_{id}$ = Unique thread identifier
1  Define $BlockD$ array of 512 floats in shared memory.
2  Define $BlockMax$ array of $Nblocks_{\Delta t}$ floats in global memory.
3  $BlockD(tb_{id}) = D_{max}(t_{id})$.                      -Each thread takes a
4  Initialize $currentThreads = 512$.                  different $D_{max}$ value-
   while $currentThreads > 1$ do
5      $halfPoint = currentThreads/2$.
       if $tb_{id} < halfPoint$ then
6          $tb2_{id} = tb_{id} + halfpoint$.
           if $BlockD(tb2_{id}) > BlockD(tb_{id})$ then
7              $BlockD(tb_{id}) = BlockD(tb2_{id})$
8      Threads synchronization.
9      $currentThreads = halfPoint$.
10 $BlockMax(b_{id}) = BlockD(0)$.                    -The maximum block value
11 Threads synchronization.                         is stored in position 0-
   if $t_{id} = 0$ then
12     Find maximum values among all the $BlockMax$ values.
13     Calculate time step $\Delta t$ with (5.3).
```

In order to obtain the maximum value in a parallel execution, an array of 512 positions is allocated in the shared memory of GPU. According to section 2.3.3, this memory is very fast and only threads within the same block can access to it. Hence, the essential strategy is that one half of the threads within each block compares two different values of D_{max} and the maximum value is stored in the shared memory, obtaining 256 values. Then, the same process is applied to these values and so on, until the maximum value within a block of threads is found. Once the maximum value of every block is obtained, only one thread is in charge of finding the absolute maximum value among those values, getting finally the maximum value used for Δt calculation. This procedure is explained in algorithm 31. Notice that this procedure requires a synchronization operation of every execution thread to ensure the comparisons are performed with updated values, which is directly accomplished with the CUDA instruction *syncthreads()*.

For its part, the evolution loop (i.e. step 11) is divided in five kernels, which are analogous to those commented in section 4.4.2.3 for the wet etching simulator. These kernels are detailed in algorithms 32, 33, 34, 35 and 36. All these operations are performed by every execution thread with a unique thread identifier t_{id}, which is in charge of evaluating a certain grid point and it is used for accessing the main variables of table 5.1.

Algorithm 32: Kernel 4 of GPU SFM implementation of profile evolution module. It corresponds to steps 12 and 13 of algorithm 28.

```
if $state\_aux(t_{id}) = 0$ then
1   Calculate spatial derivatives $\phi_q^{\pm}$.
2   According to $D_q$, apply the upwind differencing algorithm 1 to determine the
    proper derivatives, backward or forward, in each dimension $q$.
3   Store the corresponding derivatives in variables $\phi_q$ respectively.
```

Algorithm 33: Kernel 5 of GPU SFM implementation of profile evolution module. It corresponds to steps 14-16 of algorithm 28.

```
if $state\_aux(t_{id}) = 0$ then
1   Update $\phi(t_{id})$ using (5.2).
    if $\phi(t_{id}) < -0.5\Delta x$ then
2       $state\_aux(t_{id}) = -11$.
    if $\phi(t_{id}) > 0.5\Delta x$ then
3       $state\_aux(t_{id}i) = 11$.
```

Algorithm 34: Kernel 6 of GPU SFM implementation of profile evolution module. It corresponds to step 17 of algorithm 28.

```
if $state\_aux(t_{id}) = 1$ then
1   Among the six neighbouring points of $t_{id}$, find the point with minimum distance
    $\phi(t_{idB})$ value and $state = 0$.
    if no point with $state = 0$ is found then
2       $state\_aux(t_{id}) = 2$.
3       $state(t_{id}) = 2$.
    else
4       Update $\phi(t_{id}) = \phi(t_{idB}) + \Delta x$.
5       if $\phi(t_{id}) \in [-0.5\Delta x, 0.5\Delta x]$ then $state\_aux(t_{id}) = 10$.
6       if $\phi(t_{id}) > 1.5\Delta x$ then $state\_aux(t_{id}) = 2$.
if $state\_aux(t_{id}) = -1$ then
7   Among the six neighbouring points of $t_{id}$, find the point with maximum distance
    $\phi(t_{idB})$ value and $state = 0$.
    if no point with $state = 0$ is found then
8       $state\_aux(t_{id}) = -2$.
9       $state(t_{id}) = -2$.
    else
10      Update $\phi(t_{id}) = \phi(t_{idB}) - \Delta x$.
11      if $\phi(t_{id}) \in [-0.5\Delta x, 0.5\Delta x]$ then $state\_aux(t_{id}) = 10$.
12      if $\phi(t_{id}) < -1.5\Delta x$ then $state\_aux(t_{id}) = -2$.
```

Algorithm 35: Kernel 7 of GPU SFM implementation of profile evolution module. It corresponds to step 18 of algorithm 28.

```
  if $state\_aux(t_{id}) = 10$ then
1     $state(t_{id}) = 0$.
2     $state\_aux(t_{id}) = 0$.
  if $state\_aux(t_{id}) = 11$ then
3     $state(t_{id}) = 1$.
4     $state\_aux(t_{id}) = 1$.
  if $state\_aux(t_{id}) = -11$ then
5     $state(t_{id}) = -1$.
6     $state\_aux(t_{id}) = -1$.
```

Algorithm 36: Kernel 8 of GPU SFM implementation of profile evolution module. It corresponds to step 19 of algorithm 28.

```
  if $state\_aux(t_{id}) = 0$ then
      for neighbouring points $(t_{idB})$ with $state = 2$ do
1         $state(t_{idB}) = 1$.
2         $state\_aux(t_{idB}) = 1$.
3         $\phi(t_{idB}) = \phi(t_{id}) + \Delta x$.
      for neighbouring points $(t_{idB})$ with $state = -2$ do
4         $state(t_{idB}) = -1$.
5         $state\_aux(t_{idB}) = -1$.
6         $\phi(t_{idB}) = \phi(t_{id}) - \Delta x$.
```

The workflow of this GPU SFM-based implementation of the profile evolution module is presented in Fig. 5.9. As can be observed, in this implementation data transfers between CPU and GPU devices are performed only at the beginning and at the end of the evolution profile loop. Once the surface is adapted to Anetch P points, convergence is reached and the ϕ values are transferred from the GPU to the CPU in order to proceed with the extraction module.

5.3 Extraction module

As depicted in workflow diagram of Fig. 5.2, after the LS evolution module updates the surface, the final SDF is generated and the explicit surface has to be extracted to create faces and vertices suitable for Anetch. An example of the different steps of this module is shown in Fig. 5.10. The LS module only considers those points of the actual material being etched that must be updated to save memory and to reduce computational cost (see Fig. 5.10(a)). Thus, after extracting the evolved

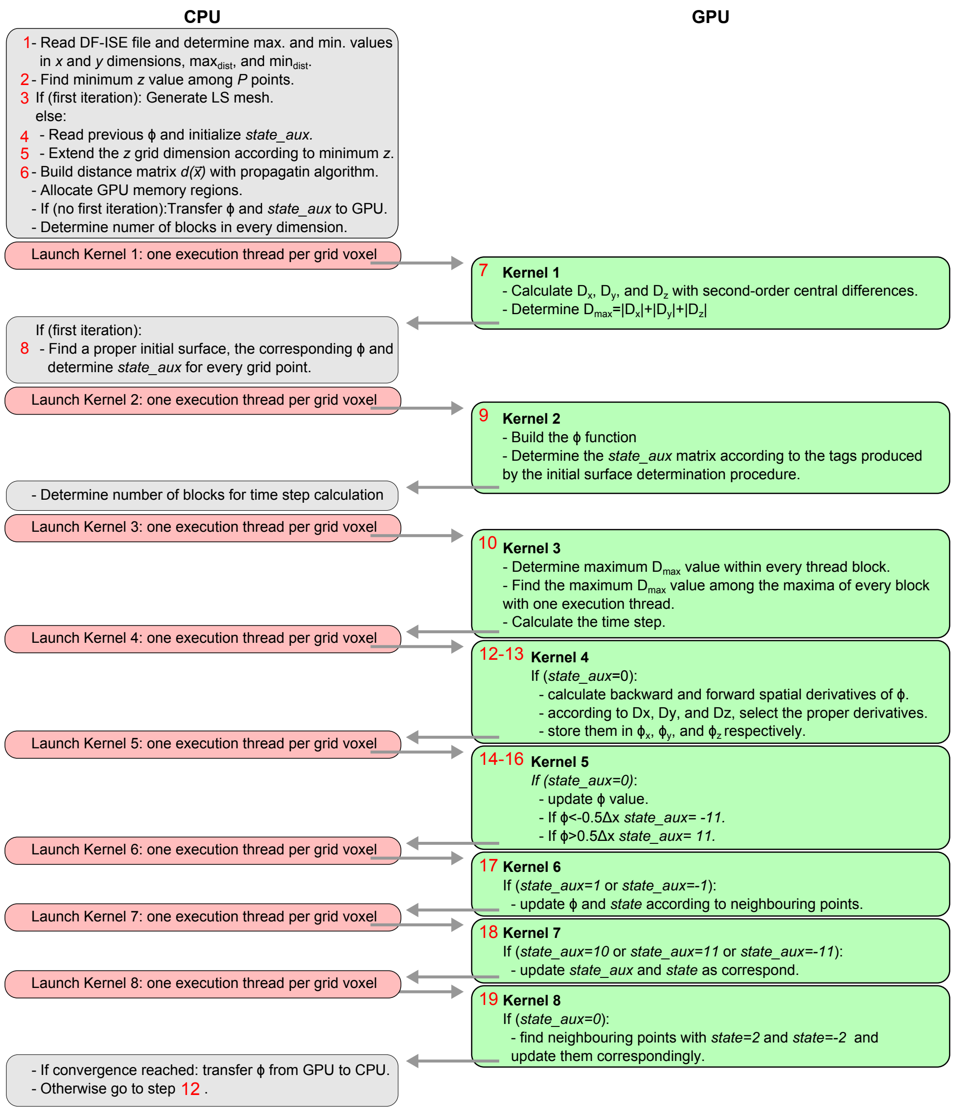

CPU
GPU
1- Read DF-ISE file and determine max. and min. values in *x* and *y* dimensions, max_{dist}, and min_{dist}.
2- Find minimum *z* value among *P* points.
3 If (first iteration): Generate LS mesh.
else:
4 - Read previous ϕ and initialize *state_aux.*
5 - Extend the *z* grid dimension according to minimum *z*.
6- Build distance matrix $d(\overline{x})$ with propagatin algorithm.
- Allocate GPU memory regions.
- If (no first iteration):Transfer ϕ and *state_aux* to GPU.
- Determine numer of blocks in every dimension.
Launch Kernel 1: one execution thread per grid voxel
7 Kernel 1
- Calculate D_x, D_y, and D_z with second-order central differences.
- Determine $D_{max}=|D_x|+|D_y|+|D_z|$
If (first iteration):
8 - Find a proper initial surface, the corresponding ϕ and determine *state_aux* for every grid point.
Launch Kernel 2: one execution thread per grid voxel
9 Kernel 2
- Build the ϕ function
- Determine the *state_aux* matrix according to the tags produced by the initial surface determination procedure.
- Determine number of blocks for time step calculation
Launch Kernel 3: one execution thread per grid voxel
10 Kernel 3
- Determine maximum D_{max} value within every thread block.
- Find the maximum D_{max} value among the maxima of every block with one execution thread.
- Calculate the time step.
Launch Kernel 4: one execution thread per grid voxel
12-13 Kernel 4
If (*state_aux*=0):
- calculate backward and forward spatial derivatives of ϕ.
- according to Dx, Dy, and Dz, select the proper derivatives.
- store them in ϕ_x, ϕ_y, and ϕ_z respectively.
Launch Kernel 5: one execution thread per grid voxel
14-16 Kernel 5
If (state_aux=0):
- update ϕ value.
- If ϕ<-0.5Δx *state_aux= -11.*
- If ϕ>0.5Δx *state_aux= 11.*
Launch Kernel 6: one execution thread per grid voxel
17 Kernel 6
If (*state_aux=1* or *state_aux=-1*):
- update ϕ and *state* according to neighbouring points.
Launch Kernel 7: one execution thread per grid voxel
18 Kernel 7
If (*state_aux=10* or *state_aux=11* or *state_aux=-11*):
- update *state_aux* and *state* as correspond.
Launch Kernel 8: one execution thread per grid voxel
19 Kernel 8
If (*state_aux=0*):
- find neighbouring points with *state=2* and *state=-2* and update them correspondingly.
- If convergence reached: transfer ϕ from GPU to CPU.
- Otherwise go to step 12 .

Figure 5.9: Parallel GPU SFM algorithm of the profile evolution module for RIE simulation. Red number indicate the equivalent step of algorithm 28.

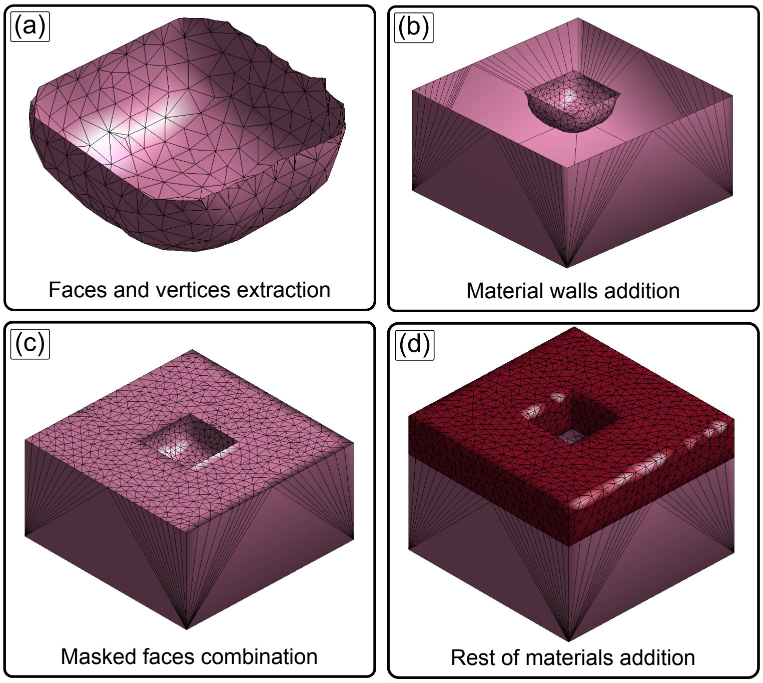


Figure 5.10: Example of the extraction process: (a) faces and vertices generation, (b) the side and bottom walls are added to the etched surface, (c) the etched surface is combined with the mask region, and (d) the rest of the materials are added to the final structure.

surface, the side and bottom walls of the substrate need to be generated and joined to the new evolved surface (step (b)). Moreover, for a coherent structure representation, those faces at interfaces between different materials must be shared by both of them. Therefore, the mask faces in contact with the etched region are combined with the rest of the faces of the etched material (step (c)). Finally, the rest of materials can be perfectly combined forming the final structure (see Fig. 5.10(d)).

5.3.1 Implementation

This module has been implemented in Matlab programming language and the input data are the DF-ISE file resulting from the previous iteration (or the original structure if it is the first iteration) and the ϕ function produced by the LS evolution profile module.

The first step of this module is to extract the surface from the LS function. To accomplish this task, the *isosurface* Matlab function is employed in the first place. This function finds those grid points that are close to the zero-level and creates vertices and faces according to the values of those points. *Isosurface* function usually generates too many faces to obtain reasonable low execution times, thus, the Matlab function *reducepatch* could be used. However, better results have been obtained by using the meshing tool presented by Fang et al. [308], since a list of vertices and faces that form a more regular triangular mesh is generated. In particular the *meshresample* function is used, which uses the mesh CGAL simplification tool that reduces the number of triangles while keeping the overall shape of the surface as much as possible [309]. This tool is based on the halfedge-collapse operation, i.e. given an edge e connecting vertices v and w, v is pulled into w, disappearing e and v, and leaving w. This operation is repeated over the whole surface until the desired number of faces is reached. A maximum number of 3000 faces has been chosen, which ensures enough accuracy for all the tested examples and reasonable low execution times are obtained. Therefore, by combining these two methods, a list of vertices and faces is generated forming a surface as can be visualized in Fig. 5.10(a).

After the explicit surface generation, this regions must be connected and combined with the rest of regions. Hence, the previous non-updated structure is loaded from the corresponding DF-ISE file. Then, the etched regions are removed and must be replaced by the updated ones. To ensure a coherent structure representation, the border vertices of the extracted surface are approximate to their closest border vertices of the adjacent regions. This allows a perfect match with the shared faces of the interface.

After this and according to the etched region dimensions loaded from the DF-ISE file, side and bottom walls are generated considering the border vertices of the interface between etched material and the rest of regions. Notice that very simple meshes can be used for defining the side walls of the etched material since these parts of the substrate are not directly modified by the etching process. The inclusion of side and bottom walls of the etched substrate can be visualized in Fig. 5.10(b).

The extraction process here commented generates a list of vertices and faces formed by three vertices each. Nevertheless, the DF-ISE format requires structures defined by vertices, edges and faces. Hence, the etched substrate is then transformed to

DF-ISE format by properly generating a list of edges defined by the connected vertices of the faces and redefining these faces according to the new generated edges. Now, interface faces can be included in the etched region. A structure with those faces shared with the mask material is represented in Fig. 5.10(c).

Finally, the rest of the regions are included in the updated structure file directly from the DF-ISE input file, producing the complete updated structure as can be shown in Fig. 5.10(d). All this procedure is summarized in algorithm 37.

Algorithm 37: Implementation of the extraction module.

```
Data: LS function $\phi$ with the update etched surface
Data: DF-ISE file describing the non-updated structure

1 Load $\phi$ function and apply isosurface MATLAB function to produce explicit faces and
  vertices.
2 Reduce the number of faces to a maximum of 3000 by using meshresample function.
3 Load DF-ISE file.
4 Approximate border vertices of the extracted surface to the closest border vertices of the
  adjacent regions for a coherent representation.
5 Generate side and bottom walls of the etched substrate.
6 Transform the updated extracted surface into DF-ISE format.
7 Add interface faces shared with other regions to the etched one.
8 Include the rest of the regions directly from the input DF-ISE file and generate the final
  updated structure.
```

5.4 Results

This section presents a collection of several simulation results. The purpose of this section is to prove the versatility and the capabilities of the developed modules to handle different kinds of topologies and produce realistic results. First, a total of four different structures are simulated to demonstrate the advantages of the implicit representation in comparison with the explicit representation employed by Anetch. Finally, a real experiment is simulated and both results are compared.

Furthermore, the execution time of each module is presented, as well as the contribution of the different parts of the LS GPU module. All the simulations have been performed on a machine consisting of an Intel Xeon CPU at 2.1 GHz with 32 GB of RAM, using a 64-bit Ubuntu-based (version 14.04 LTS) and an Nvidia GeForce GTX Titan. For the extraction module, the MATLAB 8.3 version (R2014a) has been used. Likewise, the Anetch algorithm is written in pure sequential C++ programming language whereas the SFM algorithm is implemented in parallel CUDA C.

Conf.	F	CF	CF_2	C_2F_3	C_2F_4	CF_2^+	CF_3^+	$C_2F_4^+$	$C_2F_5^+$
A	1.21e5	2.28e4	5.58e4	9.5	1.02e2	3.48e3	9.6e3	4.56	6.9
B	1.21e5	2.28e4	5.58e4	9.5	1.02e2	8.7e3	2.4e4	1.14e2	17.25

Table 5.2: Input parameters configurations: relative field values of fluxes of the different species, neutrals and ions.

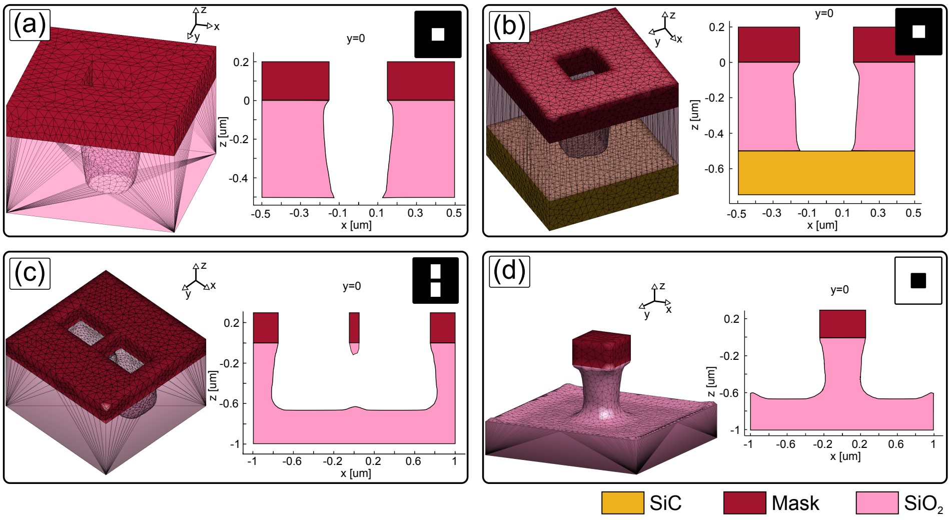


Figure 5.11: RIE simulation results containing a three-dimensional view of the final structure, a cross section and the used mask patterns.

As commented in section 5.1.1, Anetch takes as input parameters the relative field values of the considered neutrals and ions species. By modifying this parameter values, the etching behaviour is changed. Accordingly, for the simulated benchmarks, two different configurations of these values have been utilized. The field values of both configurations are collected in table 5.2.

Fig. 5.11 shows the simulations results of the tested SiO_2 etching processes. Each result contains a three-dimensional visualization of the whole structure, a cross section that provides a better understanding of the result and, in the upper right corner of each sub-figure, the used mask pattern. In addition, table 5.3 collects the relevant information of each simulation such as, first column: name of simulation, second column: size of the substrate, third column: surface dimensions in voxels of initial interface between mask and SiO_2 materials, fourth column: number of iterations of the whole algorithm shown in Fig. 5.2 and parameters configuration of table 5.2 employed, and fifth column: simulation time of each module.

Figure	Substrate size (μm)	Surf. size (voxels)	Iter. /conf.	Simulation time (s) Anetch	LS	Extrac.	Total
5.11(a)	1x1x0.7	216x216	11/A	1326	75	216	1617
5.11(b)	1x1x0.95	270x270	11/A	2255	77	373	2705
5.11(c)	2x2x1.2	322x322	14/B	9206	102	410	9718
5.11(d)	2x2x1.2	306x306	14/B	469	92	318	879
5.12	21.1x21.1x12	346x346	15/A	1032	82	585	1699

Table 5.3: Parameters of the simulated RIE processes.

Notice that in these simulations, the mask regions are considered ideal so they are not modified. Simulation (a) of Fig. 5.11 is a through hole simulation with a simple hole in the middle of the mask. Likewise, simulation (b) is the same structure with a substrate of silicon carbide at the bottom. These results show that the developed simulator consisting of Anetch, the developed SFM evolution profile module and the extraction module, is able to remove completely the etched material, as well as to stop the etching process when a different material is reached. Conversely, as observed in Fig. 5.1(a), the explicit surface representation used by the original Anetch is not able to split the surface in a similar example when reaching a different material neither completely removing the etched material at the bottom. Both simulations use the parameters of configuration A, collected in table 5.2, as input data for Anetch models.

In simulations (c) and (d), the parameters of configuration B are used. This configuration provides a higher underetching. Simulation (c) consists in two near rectangular holes in the mask. Due to the high underetching, the SiO_2 between both openings is almost completely removed. This simulation shows the capability of the developed SFM module to handle the collision of two evolving surfaces, which usually is a complex task when using explicit surfaces. As shown in Fig. 5.1(b), the original Anetch did not handled correctly this phenomenon and additional subroutines would be necessary due to the explicit surface representation employed.

Finally, simulation (d) shows the effect of the high underetching in the micromachining process of a high aspect ratio column. This RIE process is interesting since initial surface has many faces and many other new faces must be added as the surface is evolved. This phenomenon is well simulated by the implemented implicit surface representation by means of the SFM. Nevertheless, the operations of adding new faces are usually complicated for explicit representation algorithms and unrealistic results can be produced as shown in Fig. 5.1(c).

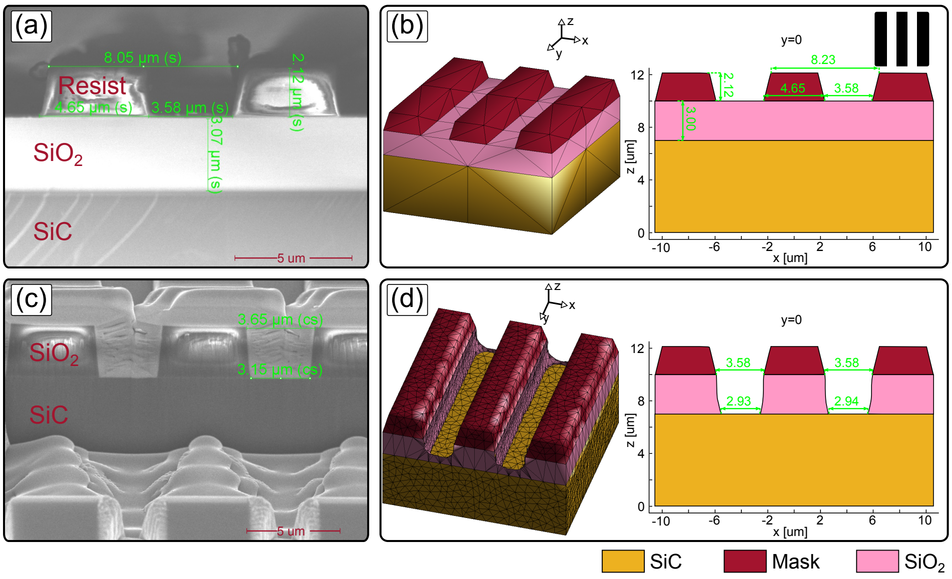


Figure 5.12: Simulation results containing a three-dimensional view of the final structure, a cross section representation, and the used mask patterns.

Finally, a comparison of experimental with simulated results is shown to validate the program. The experiment consist of a SiO_2 substrate masked with 2.12 μm thick deposited AZ5214 resist to, subsequently, form micro trenches in the substrate by fluorocarbon plasma etching. Additionally, the SiO_2 substrate is placed on a silicon carbide substrate to stop the etching process. The new formed SiO_2 trenches could be used as masking material for a subsequent SiC etching process.

The first step of the experiment is to deposit the resist on the SiO_2 substrate and bake it in the oven. The result of this step is shown in Fig. 5.12(a). This structure is considered by the simulator as the initial structure to be etched and is represented in Fig. 5.12(b). In both figures several measurements are shown for comparison. After 70 minutes of etching, with an etch rate of approximately 50 nm/min the trenches are formed. The experimental result is shown in Fig. 5.12(c).

Regarding the simulation process, the parameters used as input data correspond to the configuration A shown in table 5.2. Additionally, the simulation parameters used in this experiment are shown in the last row of the table 5.3. Fig. 5.12(d) shows the three-dimensional final structure as well as the cross section in the middle of the substrate. Notice that, despite the simulator considers an ideal resist, in the experiment the resist is also etched, resulting in $1.74\mu m$ of thickness.

According to the results, despite the side walls of the experimental trenches are straighter than the simulated ones, both structures are very similar. Every simulated measurement is in good agreement with the experimental one, differing at maximum by $0.22\mu m$ while the length of the substrate is $21.5\mu m$. Thus, the developed simulator has emulated correctly the experimental etching process.

Especially interesting is the improvement obtained with the LS module which enabled the splitting of the etched surface. The surfaces of both trenches are evolved according to Anetch models, then, when the fronts reach the SiC substrate, the corresponding ϕ values are updated taking positive values and, thus, removing the zero-level at those parts of the grid without any additional computational effort. Therefore, the simulated material is completely removed as in the experiment. On the other hand, this operation cannot be completely performed with explicit parametric surface representation as shown in example (a) of Fig. 5.1. Although this is a simple experiment, it is used for proving the efficacy of the LS method for evolving the surface in combination with etching models that use explicit surface representation.

The grid resolution of each simulation shown in table 5.3 has been chosen to obtain the results with a reasonable accuracy. According to the computational times shown in this table, the SFM module does not only depends on the grid size but, also, on the surface topography due to the SFM. For instance, despite the last example, shown in Fig. 5.12, has a larger surface grid than example (c) of Fig. 5.11, the time required by the last is higher since the active surface has more points. In every example, the LS module only represents a small part of the total computational time of the simulation (between 1.1% and 10.5%) due to the parallel CUDA-based implementation. On the other hand, the extraction module represent a higher part of the simulation (between 4.2% and 36.2%) and it depends directly on surface grid size.

In order to analyse the computational performance of the profile evolution GPU SFM implementation, the main parts of algorithm 28 are timed on the last iteration of every tested simulation process. The contributions of these main parts are presented in Fig. 5.13. As can be observed, the most consuming part of the algorithm is clearly the calculation of the matrix distance with the propagating algorithm, which takes about 50% of the total simulation time. The second part that takes more time (between 34% and 40%) is the operation of reading the ϕ function of the previous iteration. On the contrary, all the steps executed on the GPU side, like the evolution loop which includes kernels 4 to 8 (algorithms 32 to 36) takes less than 0.1% and they are not even displayed at the graphs. The reason why the GPU part takes such a little time in comparison with the steps executed on the CPU side is the high performance provided by the Nvidia GeForce

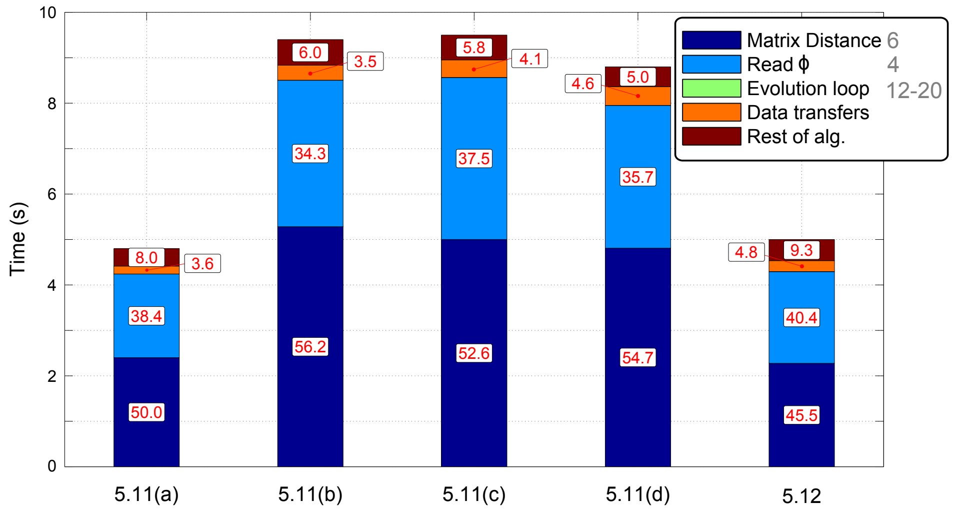


Figure 5.13: Representation of the execution times obtained with the GPU SFM implementation of the evolution profile module of the RIE simulator. The red numbers indicate the contribution of the main parts of algorithm 28. The gray numbers indicate the corresponding step of the algorithm. All the values are obtained for the last iteration of the simulation process. The contribution of the evolution is not represented since represents less than 0.1% of the simulation time.

GTX Titan in combination with the developed one thread per grid point strategy. Moreover, the CPU code is executed sequentially and the employed Xeon processor works only at 2.1 GHz.

Finally, due to the number of transfers between CPU and GPU has been minimized, they only takes 3.5 – 4.8% of the total time. Similarly, the rest of the algorithm, including kernel 1 (algorithm 29), kernel 3 (algorithm 31), and the reading of the DF-ISE file (step 1), only takes 5.0 – 9.3% of the simulation time.

5.5 Conclusions

In this chapter, the evolution profile technique used by RIE simulators is improved by means of implicit surface representation. In particular, a new module based on the LS method is developed to evolve surface being etched according to the models of Anetch, which originally utilized an explicit parametrization technique, producing unrealistic results for some etching processes.

The LS module is based on the SFM to provide accurate and fast results. Moreover, this module has been implemented in CUDA C programming language to take advantage of the parallel nature of the LS reducing the computational time. Due to the new features provided by the Nvidia GeForce GTX Titan employed in this chapter, a new strategy has been developed for updating the surface. Accordingly, one execution thread per LS mesh point is created, achieving negligible execution times for the code executed on the GPU in comparison with the operations executed on the CPU device.

In addition, another new module is implemented to extract the embedded LS surface and to join it with the rest of the structure, enabling the coupling of Anetch and the LS module. Hence, a versatile simulator of silicon dioxide etching in fluorocarbon plasma process that takes advantage of the LS features, such as the trivial handling of the interaction of several surfaces, is developed. This enables the simulation of complex processes allowing the application of the simulator to the design of MEMS.

Several simulated structures are presented, proving the versatility of the developed simulator. Different scenarios have been simulated, such as the collision of two etching surfaces due to a high underetching, the formation of a through hole in a substrate, the capability of stopping the etching process when a new material is reached, and the formation of high aspect ratio columns. The developed modules of the program coupled with Anetch models simulated correctly all the tested scenarios.

Additionally, a comparison with an experimental result is presented. The experiment consist of the formation of micro trenches in a silicon dioxide substrate. Despite the side walls of the experimental trenches are straighter than the simulated ones, both results are very similar and the measurement presented for both cases are in good agreement, even the surface has been split when reaching the other material substrate at the bottom. This behaviour emulates correctly the experimental process, proving the efficacy of the developed silicon dioxide plasma etching simulator.

Finally, the computational time of each program module is presented, proving that the LS and the extraction modules represent a small part of the simulation even when large grids are chosen, due to the parallel CUDA-based implementation. Also, computational time depends on the number of points of the evolving surface because, in the SFM, only the nearest points to the surface are updated, contrary to original LS method that computes every grid point. Moreover, the contribution of the main parts of the GPU SFM algorithm are analysed, concluding that the most of the time is taken by the CPU time due to the high performance provided by the GPU device and the corresponding implementation.

Chapter 6

Conclusions and future work

The conclusions of the different parts of this document are compiled in this chapter. Contributions derived from this research work as well as possible future work are also presented.

6.1 Conclusions

All the imposed objectives have been accomplished over the different chapters of the present thesis. These objectives are oriented to improve the simulation of wet and dry etching processes by means of the LS method. Furthermore, the optimization and acceleration of the proposed algorithms formed part of these objectives too.

In chapter 2 the fundamentals of the LS method, the corresponding numerical schemes, and some optimizations such as the SFM are commented. In addition, the usage of GPU devices as massive parallel computational platforms is introduced. Furthermore, the main micromachining processes used in MEMS fabrication are presented, including a description of wet etching and RIE processes and how they can be simulated using different approaches like CAs or the LS method.

Chapter 3 addresses one of the objectives, i.e. to improve the visualization of the results of the CA-based simulators applied to wet etching simulation. Two implementations were performed: (i) the original LS method which requires updating all the grid points and the usage of a reinitialization technique to guarantee numerical stability, and (ii) a SFM-based implementation that updates only the strictly necessary grid points. After comparing both implementations results to the corresponding cloud of points of a CCA simulator which applies a visualization technique which consists in colouring the remaining atoms depending on their normal vector, the next conclusions are reached:

- The SFM-based implementation achieves more accurate results than original LS method, while requiring less computational time.
- The proposed reconstruction implementation is able to produce smooth and continuous surfaces for all the tested structures, independently on the complexity of these topologies.
- The visualization with the proposed implementation has been clearly improved in comparison with the currently used technique.

Chapter 4 is focused on the simulation of wet etching process by means of the LS method. First, the LS is adapted to use directly experimental etch rates without regarding substrate material or crystallographic structure. The proposed algorithm is tested and validated, accomplishing one of the imposed objectives. Then, this SFM-based algorithm is implemented in two parallel versions: a multi-core CPU and a massively parallel GPU version. These parallel SFM implementations fulfil another objective. Finally, many experiments with different conditions (etchant solution, time of etching, material substrate, etc.) are simulated and the results obtained with the proposed algorithm are compared to the CCA corresponding ones as well as to those experimental. This thorough comparison satisfies another objective. Accordingly, the conclusions of this chapter can be summarized as follows:

- The LS method is capable of simulate accurately all the tested etchant solutions (including different concentrations and temperatures of KOH, KOH+IPA, TMAH, TMAH+Triton, and NH_4HF_2) and substrate materials (silicon and quartz) without the need for recalibrating the internal parameters, contrary to the state-of-the-art CCA simulators.
- The proposed SFM implementation produces less noisy results than CCA approach.
- SFM approach produces similar accuracy as CCA, although for highly anisotropic etchants, the SFM tends to soften corners and edges.
- When comparing pure sequential implementations of the SFM and the CCA approach, the latter is up to 2 times faster than the SFM for anisotropic etchants, whereas the SFM can be up to 10 times faster than CCA for isotropic etchants due to the simplification that can be applied to the evolution equation of the LS. In addition, SFM usually requires roughly 4 times less memory.

- The SFM has been satisfactory implemented in two parallel computing platforms: multi-core CPU and many-core GPU. This enables to obtain simulation times of just a few seconds.
- GPU implementation of the SFM algorithm is faster than parallel CPU version.
- When comparing GPU implementations of the SFM and the CCA approach, the latter is usually slower than the SFM. However this is assigned to the use of an octree data structure in the CCA method that requires additional calculations and managements operations in order to reduce memory allocation.

Finally, in chapter 5 the last objective is accomplished, i.e. to improve current RIE simulators that use surface explicit representation, by developing a LS-based tool which takes advantage of implicit representation and enables the possibility of simulating more complex experiments. Accordingly, a GPU implementation of the proposed SFM-based algorithm which takes advantage of the new features of Nvidia devices is developed. This implementation has been perfectly coupled to Anetch RIE simulator by developing a module that transform the surface from its implicit LS representation to the corresponding explicit one. The next points can be conclude:

- The proposed reconstruction algorithm enables the simulation of complex processes such as through holes and coalescing of multiple surfaces.
- The proposed SFM implementation can be perfectly couple to RIE models that use explicit surface representation like Anetch.
- The resulting structures obtained with the developed tools are realistic, contrary to those obtained with explicit representation of Anetch.
- The complete developed simulator is validated by simulating a real experiment. Both results are in good agreement, although the side walls obtained in the experiment are straighter.
- Due to modern Nvidia GPU devices, the parts of the parallel SFM algorithm executed on GPU side are drastically accelerated, such that the LS module of a whole simulation only takes a small part of total time.

Although the study of this chapter is focused on the silicon dioxide RIE process, different models could be used in combination to the developed SFM and extraction modules.

6.2 Contributions

The research work of the present thesis has made possible the next scientific contributions:

- Article publication in the conference: *Mathematical Modelling in Engineering & Human Behaviour 2012*. In this article, the proposed SFM implementation for surface reconstruction applied to improve the visualization of the CCA simulation results is presented. Some resulting structures are included and compared to CA results. This article was later published in the *International Journal of Computer Mathematics*, listed on the JCR with an impact factor of 0.721 (2013).
  C. Montoliu, N. Ferrando, J. Cerdá, and R. J. Colom, *Application of the level set method for the visual representation of continuous cellular automata oriented to anisotropic wet etching*, International Journal of Computer Mathematics, vol. 91, no. 1, pp. 124-134, 2014.

- Article publication in: *Computer Physics Communications*, listed on the JCR with an impact factor of 2.407 (2013). This article describes the first SFM proposed algorithm for wet etching simulation, including the parallel CPU and GPU implementations. In addition, simple etching processes are used as benchmarks to compare the sequential implementations of the SFM and the CCA approach.
  C. Montoliu, N. Ferrando, M. Gosálvez, J. Cerdá, and R. Colom, *Implementation and evaluation of the Level Set method: Towards efficient and accurate simulation of wet etching for microengineering applications*, Computer Physics Communications, vol. 184, no. 10, pp. 2299-2309, 2013.

- Article publication in: *Journal of Micromechanics and Microengineering*, listed on the JCR with an impact factor of 1.725 (2013). In this article, the GPU implementation capable of simulating complex wet etching processes was presented. An extensive set of silicon and quartz based experiments were simulated and compared to a CCA approach as well as to the experimental results.
  C. Montoliu, N. Ferrando, M. Gosálvez, J. Cerdá, and R. Colom, *Level Set implementation for the simulation of anisotropic etching: application to complex MEMS micromachining*, Journal of Micromechanics and Microengineering, vol. 23, no. 7, p. 075017, 2013.

- Article publication in: *Journal of Micromechanics and Microengineering*, listed on the JCR with an impact factor of 1.725 (2013). The proposed LS and extraction modules for RIE simulation are detailed, including a comparison with an experimental result.
  C. Montoliu, E. Baer, J. Cerdá and R. J. Colom, *Improvement of Feature-Scale Profile Evolution in Silicon Dioxide Plasma Etching Simulator by Using*

the Level Set Method, Journal of Micromechanics and Microengineering, vol. 25, no. 6, p. 065013, 2015.

6.3 Future work

Despite the solutions proposed in the present thesis have provided a significant progress in the field of LS efficient simulation and, specially, applied to micromachining processes emulation, there is still much room for further research. Several lines of work can be originated from this research:

- To develop a GPU implementation of the reconstruction algorithm in order to obtain the improved visualizations of results in very short computational times. This would make the developed tool a more interesting method to form part of atomistic simulators, easing the understanding and the analysis of such structures.

- Despite the amount of memory required by the implemented SFM-based wet etching simulator is relatively low, the implementation could be improved by using octrees data structures to allocate only those active surface points instead of the whole LS grid. This could enable the possibility of using finer meshes to obtain more accurate results.

- Although a lot of effort has been invested in the presented GPU implementations, it is possible that the corresponding algorithms could be further debugged and optimized, thus obtaining lower simulation times.

- A more ambitious project would be to include the possibility of adding sacrificial layers of different materials on the same simulation, such that, depending on the selected etchant solution, only the corresponding materials would be removed. This would enable the simulation of creating even more complex structures formed by several materials.

- According to this, more experimental effects could be taken into account by the wet etching simulator, such as the etch-stop effect that produce highly boron-doped silicon, or the nonideal mask patterns (i.e. the slow material removal that also suffers the masks when etching the substrate). The inclusion of such phenomena would produce more realistic results.

- To test the developed modules for profile evolution in RIE simulation to other etching models. These modules were tested on silicon dioxide Anetch models, however it must be possible to obtain good results for other etching models.

- Another different topic to be researched would be to develop a RIE simulator based on the SFM similar to Anetch but capable of calculating local etch rates of surface without the need of extracting the implicit surface from the LS function. If the extraction part of the simulator could be avoid, much faster simulations would be obtained.

- In addition, would be interesting to develop a GPU implementation of the etch rate calculation part (i.e. the current Anetch module) in order to accelerate the simulations.

- Another possible research line would consist in developing new LS algorithms for the simulation of other different micromachining processes, such as vapour deposition or DRIE, which are widely used in MEMS fabrication.

All of these proposed lines of work will be addressed in the postdoctoral stage.

List of Figures

List of Tables

List of Algorithms and Procedures

Bibliography

[1] J. E. P. Jr. and E. G. Puckett, “Second-order accurate volume-of-fluid algorithms for tracking material interfaces,” *Journal of Computational Physics*, vol. 199, no. 2, pp. 465 – 502, 2004.

[2] Y. Renardy and M. Renardy, “Prost: A parabolic reconstruction of surface tension for the volume-of-fluid method,” *Journal of Computational Physics*, vol. 183, no. 2, pp. 400 – 421, 2002.

[3] S. J. Cummins, M. M. Francois, and D. B. Kothe, “Estimating curvature from volume fractions,” *Computers & Structures*, vol. 83, no. 6, pp. 425 – 434, 2005. Frontier of Multi-Phase Flow Analysis and Fluid-Structure Frontier of Multi-Phase Flow Analysis and Fluid-Structure.

[4] G. Weymouth and D. K.-P. Yue, “Conservative volume-of-fluid method for free-surface simulations on cartesian-grids,” *Journal of Computational Physics*, vol. 229, no. 8, pp. 2853 – 2865, 2010.

[5] S. Osher and J. A. Sethian, “Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations,” *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988.

[6] D. Adalsteinsson and J. A. Sethian, “A fast level set method for propagating interfaces,” *Journal of Computational Physics*, vol. 118, no. 2, pp. 269–277, 1995.

[7] R. T. Whitaker, “A level-set approach to 3D reconstruction from range data,” *International Journal of Computer Vision*, vol. 29, pp. 203–231, 1998.

[8] H. Terashima and G. Tryggvason, “A front-tracking/ghost-fluid method for fluid interfaces in compressible flows,” *Journal of Computational Physics*, vol. 228, no. 11, pp. 4012 – 4037, 2009.

[9] T. V. Vu, G. Tryggvason, S. Homma, J. C. Wells, and H. Takakura, “A front-tracking method for three-phase computations of solidification with volume

change," *JOURNAL OF CHEMICAL ENGINEERING OF JAPAN*, vol. 46, no. 11, pp. 726–731, 2013.

[10] N. G. Deen and J. Kuipers, "Direct numerical simulation of wall-to liquid heat transfer in dispersed gas-liquid two-phase flow using a volume of fluid approach," *Chemical Engineering Science*, vol. 102, no. 0, pp. 268–282, 2013.

[11] J. López-Herrera, S. Popinet, and M. Herrada, "A charge-conservative approach for simulating electrohydrodynamic two-phase flows using volume-of-fluid," *Journal of Computational Physics*, vol. 230, no. 5, pp. 1939–1955, 2011.

[12] M. Roberts, J. Packer, M. C. Sousa, and J. R. Mitchell, "A work-efficient gpu algorithm for level set segmentation," in *Proceedings of the Conference on High Performance Graphics*, HPG '10, pp. 123–132, Eurographics Association, 2010.

[13] L. Wang and C. Pan, "Robust level set image segmentation via a local correntropy-based k-means clustering," *Pattern Recognition*, vol. 47, no. 5, pp. 1917 – 1925, 2014.

[14] M. BURGER and S. J. OSHER, "A survey on level set methods for inverse problems and optimal design," *European Journal of Applied Mathematics*, pp. 263–301, 4 2005.

[15] X.-C. Tai and H. Li, "A piecewise constant level set method for elliptic inverse problems," *Applied Numerical Mathematics*, vol. 57, no. 5-7, pp. 686–696, 2007. Special Issue for the International Conference on Scientific Computing.

[16] D. Adalsteinsson and J. Sethian, "A level set approach to a unified model for etching, deposition, and lithography i: Algorithms and two-dimensional simulations," *Journal of Computational Physics*, vol. 120, no. 1, pp. 128–144, 1995.

[17] D. Adalsteinsson and J. Sethian, "A level set approach to a unified model for etching, deposition, and lithography ii: Three-dimensional simulations," *Journal of Computational Physics*, vol. 122, no. 2, pp. 348–366, 1995.

[18] D. Adalsteinsson and J. A. Sethian, "A level set approach to a unified model for etching, deposition, and lithography iii: Redeposition, reemission, surface diffusion, and complex simulations," *Journal of Computational Physics*, vol. 138, no. 1, pp. 193–223, 1997.

[19] C. Montoliu, N. Ferrando, M. A. Gosálvez, J. Cerdá, and R. J. Colom, "Level set implementation for the simulation of anisotropic etching: application to complex mems micromachining," *Journal of Micromechanics and Microengineering*, vol. 23, no. 7, p. 075017, 2013.

[20] C. Montoliu, N. Ferrando, M. Gosálvez, J. Cerdá, and R. Colom, “Implementation and evaluation of the level set method: Towards efficient and accurate simulation of wet etching for microengineering applications,” *Computer Physics Communications*, vol. 184, no. 10, pp. 2299–2309, 2013.

[21] H.-B. Kim, G. Hobler, A. Steiger, A. Lugstein, and E. Bertagnolli, “Full three-dimensional simulation of focused ion beam micro/nanofabrication,” *Nanotechnology*, vol. 18, no. 24, p. 245303, 2007.

[22] Y. Li, D. Lee, C. Lee, J. Lee, S. Lee, J. Kim, S. Ahn, and J. Kim, “Surface embedding narrow volume reconstruction from unorganized points,” *Computer Vision and Image Understanding*, vol. 121, no. 0, pp. 100–107, 2014.

[23] C. Montoliu, N. Ferrando, J. Cerdá, and R. J. Colom, “Application of the level set method for the visual representation of continuous cellular automata oriented to anisotropic wet etching,” *International Journal of Computer Mathematics*, vol. 91, no. 1, pp. 124–134, 2014.

[24] J. A. Sethian, “Curvature and the evolution of fronts,” *Communication of Mathematical Physics*, vol. 101, no. 4, pp. 487–499, 1985.

[25] J. A. Sethian, “Evolution, implementation, and application of level set and fast marching methods for advancing fronts,” *Journal of computational physics*, vol. 169, pp. 503–555, 2001.

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

[27] J. A. Sethian, “Numerical methods for propagating fronts,” *Variational Methods for Free Surface Interfaces, edited by P. Concus and R. Finn .Springer-Verlag, New York*, 1987.

[28] M. G. Crandall and P. L. Lions, “Viscosity solutions of hamilton-jacobi equations,” *Trans. Armer. Math. Soc.*, vol. 277, pp. 1–42, 1983.

[29] M. G. Crandall, L. C. Evans, and P. L. Lions, “Some properties of viscosity solutions of hamilton-jacobi equations,” *Trans. Armer. Math. Soc*, vol. 282, no. 2, 1984.

[30] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, “Uniformly high order accurate essentially non-oscillatory schemes, III,” *Journal of Computational Physics*, vol. 131, no. 1, pp. 3–47, 1997.

[31] S. Osher and C.-W. Shu, “High order essentially non-oscillatory schemes for hamilton-jacobi equations,” *SIAM J. Numer. Anal.*, vol. 28, pp. 902–921, 1991.

[32] G.-S. Jian and D. Peng, “Weighted ENO schemes for hamilton-jacobi equations,” *SIAM Journal on Scientific Computing*, vol. 21, pp. 2126–2143, 1997.

[33] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces.* Applied Mathematical Sciences, Springer, 2003.

[34] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[35] P. Rosenthal and L. Linsen, “Direct isosurface extraction from scattered volume data,” *Eurographics/ IEEE-VGTC Symposium on Visualization*, pp. 99–106, 2006.

[36] W. Mulder and S. Osher, “Computing interface motion in compressible gas dynamics,” *Journal of Computational Physics*, vol. 100, pp. 209–228, 1992.

[37] J. Gomes and O. Faugeras, “Reconciling distance functions and level sets,” *Journal of Visual Communication and Image Representation*, vol. 11, pp. 209–223, 1999.

[38] D. L. Chopp, “Computing minimal surfaces via level set curvature flow,” *Journal of Computational physics*, vol. 106, pp. 77–91, 1993.

[39] Y.-G. Chen, Y. Giga, and S. Goto, “Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations,” *Journal Differential geometry*, vol. 33, pp. 749–786, 1991.

[40] L. C. Evans and J. Spruck, “Motion of level sets by mean curvature i,” *Journal Differential geometry*, vol. 33, pp. 635–681, 1991.

[41] B. Merriman, J. K. Bence, and S. J. Osher, “Motion of multiple junctions: A level set approach,” *Journal of Computational Physics*, vol. 112, pp. 334–363, 1994.

[42] M. Sussman, P. Smereka, and S. Osher, “A level set approach for computing solutions to incompressible two-phase flow,” *Journal of Computational Physics*, vol. 114, pp. 146–159, 1994.

[43] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, “A pde-based fast local level set method,” *Journal of Computational Physics*, vol. 155, pp. 410–438, 1999.

[44] C. Basting and D. Kuzmin, “A minimization-based finite element formulation for interface-preserving level set reinitialization,” *Computing*, vol. 95, no. 1, pp. 13–25, 2013.

[45] G. D. Rocca and G. Blanquart, “Level set reinitialization at a contact line,” *Journal of Computational Physics*, vol. 265, no. 0, pp. 34–49, 2014.

[46] D. Hartmann, M. Meinke, and W. Schröder, “The constrained reinitialization equation for level set methods,” *Journal of Computational Physics*, vol. 229, no. 5, pp. 1514 – 1535, 2010.

[47] J. A. Sethian, “A fast marching level set method for monotonically advancing fronts,” *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[48] D. Chopp, “Another look at velocity extensions in the level set method,” *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3255–3273, 2009.

[49] S. Bak, J. McLaughlin, and D. Renzi, “Some improvements for the fast sweeping method,” *SIAM Journal on Scientific Computing*, vol. 32, no. 5, pp. 2853–2874, 2010.

[50] R. Courant, K. Friedrichs, and H. Lewy, “Über die partiellen differenzengleichungen der mathematischen physik,” *Mathematische Annalen*, vol. 100, pp. 32–74, 1928.

[51] C.-W. Shu and S. Osher, “Efficient implementation of essentially non-oscillatory shock-capturing schemes,” *Journal of Computational Physics*, vol. 77, no. 2, pp. 439–471, 1988.

[52] C.-W. Shu and S. Osher, “Efficient implementation of essentially non-oscillatory shock-capturing schemes, II,” *Journal of Computational Physics*, vol. 83, no. 1, pp. 32 – 78, 1989.

[53] G.-S. Jiang and C.-W. Shu, “Efficient implementation of weighted ENO schemes,” *Journal of Computational Physics*, vol. 126, no. 1, pp. 202 – 228, 1996.

[54] C.-W. Shu, “Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws,” in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations* (A. Quarteroni, ed.), vol. 1697 of *Lecture Notes in Mathematics*, pp. 325–432, Springer Berlin Heidelberg, 1998.

[55] X. Zhang, Y. Liu, and C. Shu, “Maximum-principle-satisfying high order finite volume weighted essentially nonoscillatory schemes for convection-diffusion equations,” *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A627–A658, 2012.

[56] W.-S. Don and R. Borges, “Accuracy of the weighted essentially non-oscillatory conservative finite difference schemes,” *Journal of Computational Physics*, vol. 250, no. 0, pp. 347–372, 2013.

[57] X. Zhong and C.-W. Shu, "A simple weighted essentially nonoscillatory limiter for Runge-Kutta discontinuous Galerkin methods," *Journal of Computational Physics*, vol. 232, no. 1, pp. 397–415, 2013.

[58] M. G. Crandall and P. L. Lions, "Two approximations of solutions of hamilton-jacobi equations," *Mathematics of Computation*, vol. 43, no. 167, pp. 1–19, 1984.

[59] C. Y. Kao, S. Osher, and J. Qian, "Lax-friedrichs sweeping scheme for static hamilton-jacobi equations," *Journal of Computational Physics*, vol. 196, no. 1, pp. 367–391, 2004.

[60] H. Jiang and W. Tong, "New lax-friedrichs scheme for convective-diffusion equation," in *Information Computing and Applications* (B. Liu, M. Ma, and J. Chang, eds.), vol. 7473 of *Lecture Notes in Computer Science*, pp. 269–276, Springer Berlin Heidelberg, 2012.

[61] W. Chen, C.-S. Chou, and C.-Y. Kao, "Lax-friedrichs fast sweeping methods for steady state problems for hyperbolic conservation laws," *Journal of Computational Physics*, vol. 234, no. 0, pp. 452–471, 2013.

[62] C. Y. Kao, S. Osher, and J. Qian, "Lax-friedrichs sweeping scheme for static hamilton-jacobi equations," *Journal of Computational Physics*, vol. 196, no. 1, pp. 367–391, 2004.

[63] S. K. Godunov, "A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics," *Matematicheskii Sbornik*, vol. 89, no. 3, pp. 271–306, 1959.

[64] M. Gage, "Curve shortening makes convex curves circular," *Inventiones mathematicae*, vol. 76, no. 2, pp. 357–364, 1984.

[65] M. E. Gage and R. S. Hamilton, "The heat equation shrinking convex plane curves," *Journal of Differential Geometry*, vol. 23, pp. 69–96, 1986.

[66] M. A. Grayson, "The heat equation shrinks embedded plane curves to round points," *Journal Differential geometry*, vol. 26, pp. 285–314, 1987.

[67] J. Sethian, *An Analysis of Flame Propagation.* University of California, Berkeley, 1982.

[68] C. Lee, J. Dolbow, and P. J. Mucha, "A narrow-band gradient-augmented level set method for multiphase incompressible flow," *Journal of Computational Physics*, vol. 273, no. 0, pp. 12–37, 2014.

[69] B. Yan, C. Li, M. Xie, and C. Davatzikos, "Narrow band region-scalable fitting model for image segmentation in the presence of intensity inhomogeneities," in *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, pp. 1994–1997, March 2011.

[70] C. Li, C. Xu, C. Gui, and M. Fox, "Distance regularized level set evolution and its application to image segmentation," *Image Processing, IEEE Transactions on*, vol. 19, pp. 3243–3254, Dec 2010.

[71] Y. Shi and W. C. Karl, "A fast level set method without solving PDEs," in *ICASSP (2)*, pp. 97–100, 2005.

[72] J. Malcolm, Y. Rathi, A. Yezzi, and A. Tannenbaum, "Fast approximate surface evolution in arbitrary dimension," in *Medical imaging*, pp. 69144C–69144C, International Society for Optics and Photonics, 2008.

[73] Y. Shi and W. Karl, "A real-time algorithm for the approximation of level-set-based curve evolution," *Image Processing, IEEE Transactions on*, vol. 17, pp. 645–656, May 2008.

[74] J. Strain, "Tree methods for moving interfaces," *Journal of Computational Physics*, vol. 151, no. 2, pp. 616–648, 1999.

[75] F. Losasso, F. Gibou, and R. Fedkiw, "Simulating water and smoke with an octree data structure," *ACM Trans. Graph.*, vol. 23, pp. 457–462, Aug. 2004.

[76] C. Min and F. Gibou, "A second order accurate level set method on non-graded adaptive cartesian grids," *Journal of Computational Physics*, vol. 225, no. 1, pp. 300 – 321, 2007.

[77] S. Laine and T. Karras, "Efficient sparse voxel octrees," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, pp. 1048–1059, Aug 2011.

[78] J. Papac, A. Helgadottir, C. Ratsch, and F. Gibou, "A level set approach for diffusion and stefan-type problems with robin boundary conditions on quadtree/octree adaptive cartesian grids," *Journal of Computational Physics*, vol. 233, no. 0, pp. 241 – 261, 2013.

[79] B. Houston, M. B. Nielsen, C. Batty, O. Nilsson, and K. Museth, "Hierarchical rle level set: A compact and versatile deformable surface representation," *ACM Trans. Graph.*, vol. 25, pp. 151–175, Jan. 2006.

[80] R. E. Bridson, *Computational aspects of dynamic surfaces.* PhD thesis, Stanford university, 2003.

[81] E. Brun, A. Guittet, and F. Gibou, "A local level-set method using a hash table data structure," *Journal of Computational Physics*, vol. 231, no. 6, pp. 2528 – 2536, 2012.

[82] J. Owens, M. Houston, D. Luebke, S. Green, J. Stone, and J. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, pp. 879–899, May 2008.

[83] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. KrÃ$\frac{1}{4}$ger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[84] “Nvidia GeForce GTX Titan Z datasheet.” http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-z/specifications.

[85] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, “State-of-the-art in heterogeneous computing,” *Scientific Programming*, vol. 18, no. 1, pp. 1–33, 2010.

[86] S. Aluru and N. Jammula, “A review of hardware acceleration for computational genomics,” *Design Test, IEEE*, vol. 31, pp. 19–30, Feb 2014.

[87] A. R. Brodtkorb, T. R. Hagen, and M. L. Sætra, “Graphics processing unit (GPU) programming strategies and trends in GPU computing,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 4–13, 2013. Metaheuristics on GPUs.

[88] D. Chen and D. Singh, “Fractal video compression in opencl: An evaluation of cpus, gpus, and fpgas as acceleration platforms,” in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pp. 297–304, Jan 2013.

[89] J. Fowers, G. Brown, P. Cooke, and G. Stitt, “A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA ’12, (New York, NY, USA), pp. 47–56, ACM, 2012.

[90] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, K. A. Yelick, M. J. Demmel, W. Plishker, J. Shalf, S. Williams, and K. Yelick, “The landscape of parallel computing research: A view from berkeley,” tech. rep., TECHNICAL REPORT, UC BERKELEY, 2006.

[91] “Intel Xeon E7 datasheet.” http://ark.intel.com/es-es/products/75258/Intel-Xeon-Processor-E7-8890-v2-37_5M-Cache-2_80-GHz.

[92] “Nvidia CUDA C programming guide.” http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[93] A. Meade, J. Buckley, and J. J. Collins, “Challenges of evolving sequential to parallel code: An exploratory review,” in *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL ’11, (New York, NY, USA), pp. 1–5, ACM, 2011.

[94] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), pp. 483–485, ACM, 1967.

[95] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach.* Newnes, 2012.

[96] J. Jin, S. Turner, B.-S. Lee, J. Zhong, and B. He, "Simulation of information propagation over complex networks: Performance studies on multi-gpu," in *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*, pp. 179–188, Oct 2013.

[97] B. Block, M. Lukáčová-Medvid'ová, P. Virnau, and L. Yelash, "Accelerated gpu simulation of compressible flow by the discontinuous evolution galerkin method," *The European Physical Journal Special Topics*, vol. 210, no. 1, pp. 119–132, 2012.

[98] P. Berczik, R. Spurzem, S. Zhong, L. Wang, K. Nitadori, T. Hamada, and A. Veles, "Up to 700k gpu cores, kepler, and the exascale future for simulations of star clusters around black holes," in *Supercomputing* (J. Kunkel, T. Ludwig, and H. Meuer, eds.), vol. 7905 of *Lecture Notes in Computer Science*, pp. 13–25, Springer Berlin Heidelberg, 2013.

[99] Y. Zhuge, Y. Cao, J. K. Udupa, and R. W. Miller, "Parallel fuzzy connected image segmentation on gpu," *Medical Physics*, vol. 38, no. 7, pp. 4365–4371, 2011.

[100] B. Fulkerson and S. Soatto, "Really quick shift: Image segmentation on a gpu," in *Trends and Topics in Computer Vision* (K. Kutulakos, ed.), vol. 6554 of *Lecture Notes in Computer Science*, pp. 350–358, Springer Berlin Heidelberg, 2012.

[101] C. Kauffmann and N. Piché, "Seeded nd medical image segmentation by cellular automaton on gpu," *International Journal of Computer Assisted Radiology and Surgery*, vol. 5, no. 3, pp. 251–262, 2010.

[102] N. Ferrando, M. Gosálvez, J. Cerdá, R. Gadea, and K. Sato, "Octree-based, GPU implementation of a continuous cellular automaton for the simulation of complex, evolving surfaces," *Computer Physics Communications*, vol. 182, no. 3, pp. 628–640, 2011.

[103] M. Macedonia, "The gpu enters computing's mainstream," *Computer*, vol. 36, pp. 106–108, Oct. 2003.

[104] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for gpus: stream computing on graphics hardware," in *ACM Transactions on Graphics (TOG)*, vol. 23, pp. 777–786, ACM, 2004.

[105] M. McCool, S. Du Toit, T. Popa, B. Chan, and K. Moule, "Shader algebra," *ACM Transactions on Graphics*, vol. 23, pp. 787–795, Aug. 2004.

[106] M. D. McCool, "Data-parallel programming on the Cell BE and the GPU using the RapidMind development platform," in *GSPx Multicore Applications Conference*, vol. 9, 2006.

[107] J. Stone, D. Gohara, and G. Shi, "OpenCL: a parallel programming standard for heterogeneous computing systems," *Computing in Science Engineering*, vol. 12, pp. 66–73, May 2010.

[108] A. Shan, "Heterogeneous processing: a strategy for augmenting moore's law," *Linux Journal*, vol. 2006, no. 142, p. 7, 2006.

[109] D. P. Singh, T. S. Czajkowski, and A. Ling, "Harnessing the power of fpgas using altera's opencl compiler," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, (New York, NY, USA), pp. 5–6, ACM, 2013.

[110] "Technical brief NVIDIA GeForce GTX 200 GPU architectural overview." http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf.

[111] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pp. 235–246, March 2010.

[112] "NVIDIA's next generation CUDA compute architecture: Kepler gk110." http://www.nvidia.com/content/PDF/kepler/NVIDIA-kepler-GK110-Architecture-Whitepaper.pdf.

[113] "NVIDIA GeForce GTX 560 GPU specifications." http://www.nvidia.es/object/product-geforce-gtx-560-es.html.

[114] "NVIDIA GeForce GTX 660 GPU specifications." http://la.nvidia.com/object/geforce-gtx-660-la.html#pdpContent=2.

[115] G. Pagès and B. Wilbertz, "Gpgpus in computational finance: massive parallel computing for american style options," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 8, pp. 837–848, 2012.

[116] A. Gaikwad and I. Toke, "Parallel iterative linear solvers on gpu: A financial engineering case," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pp. 607–614, Feb 2010.

[117] H.-V. Dang, B. Schmidt, A. Hildebrandt, and A. Hildebrandt, "Parallelized clustering of protein structures on cuda-enabled gpus," in *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pp. 1–8, Feb 2014.

[118] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, and Y. Shi, "Parallel data mining techniques on graphics processing unit with compute unified device architecture (cuda)," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 942–967, 2013.

[119] A. Paz and A. Plaza, "GPU implementation of target and anomaly detection algorithms for remotely sensed hyperspectral image analysis," vol. 7810, pp. 78100R–78100R–10, 2010.

[120] S. Fehlmann, D. Booth, P. Janney, C. Pontecorvo, P. Aquilina, T. Scoleri, N. Redding, and R. Christie, "Application of detection and recognition algorithms to persistent wide area surveillance," in *Digital Image Computing: Techniques and Applications (DICTA), 2013 International Conference on*, pp. 1–8, Nov 2013.

[121] C. E, J. B, and O. K, "Method of preparing electrostatic shutter mosaics," 1956. US Patent 2,749,598.

[122] Y.-C. Tai, *Introduction to MEMS, in Microsystems and Nanotechnology, Chapter 6, Book Ed. Zhaoying Zhou, Zhonglin Wang and Liwei Lin*. Springer Berlin Heidelberg, 2012.

[123] O. Solgaard, A. Godil, R. Howe, L. Lee, Y.-A. Peter, and H. Zappe, "Optical MEMS: From micromirrors to complex systems," *Journal of Microelectromechanical Systems*, vol. 23, pp. 517–538, June 2014.

[124] S.-J. Park, I. Reines, C. Patel, and G. Rebeiz, "High-Q RF-MEMS 4-6 GHz tunable evanescent-mode cavity filter," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 58, pp. 381–389, Feb 2010.

[125] V. Sekar, M. Armendariz, and K. Entesari, "A 1.2-1.6 GHz substrate-integrated-waveguide RF MEMS tunable filter," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 59, pp. 866–876, April 2011.

[126] A. Morris, S. Caporal Del Barrio, J. Shin, V. Steel, and G. Pedersen, "Tunable antennas for mobile devices: Achieving high performance in compelling form factors," in *Microwave Symposium (IMS), 2014 IEEE MTT-S International*, pp. 1–4, June 2014.

[127] M. Higaki and K. Hashimoto, "Automatic tunable antenna system using mems variable capacitors and a probe triggered by tpc," in *Antennas and Propagation Society International Symposium (APSURSI), 2014 IEEE*, pp. 452–453, July 2014.

[128] M. de Jongh, A. van Bezooijen, K. Boyle, and T. Bakker, "Mobile phone performance improvements using an adaptively controlled antenna tuner," in *Microwave Symposium Digest (MTT), 2011 IEEE MTT-S International*, pp. 1–4, June 2011.

[129] M. Daneshmand and R. Mansour, "Rf mems satellite switch matrices," *Microwave Magazine, IEEE*, vol. 12, pp. 92–109, Aug 2011.

[130] B. Schoenlinner, A. Stehle, C. Siegel, W. Gautier, B. Schulte, S. Figur, U. Prechtel, and V. Ziegler, "The low-complexity RF MEMS switch at EADS: an overview," *International Journal of Microwave and Wireless Technologies*, vol. 3, pp. 499–508, 10 2011.

[131] G. Exposito-Dominguez, J.-M. F. Gonzalez, P. P. de la Torre, and M. Sierra-Castaner, "Dual circular polarized steering antenna for satellite communications in x band," *Progress In Electromagnetics Research*, vol. 122, pp. 61–76, 2012.

[132] M. M. G. Grafton, T. Maleki, M. D. Zordan, L. M. Reece, R. Byrnes, A. Jones, P. Todd, and J. F. Leary, "Microfluidic MEMS hand-held flow cytometer," vol. 7929, pp. 79290C–79290C–10, 2011.

[133] S. Indermun, R. Luttge, Y. E. Choonara, P. Kumar, L. C. du Toit, G. Modi, and V. Pillay, "Current advances in the fabrication of microneedles for transdermal delivery," *Journal of Controlled Release*, vol. 185, no. 0, pp. 130–138, 2014.

[134] L. Tedeschi, C. Domenici, V. Russino, A. Nannini, and F. Pieri, "Label-free detection of specific RNA sequences by a DNA-Based CMOS BioMEMS," in *Sensors and Microsystems* (C. Di Natale, V. Ferrari, A. Ponzoni, G. Sberveglieri, and M. Ferrari, eds.), vol. 268 of *Lecture Notes in Electrical Engineering*, pp. 277–280, Springer International Publishing, 2014.

[135] H. Ceylan, H. Kulah, A. Alp, C. OÌˆzgen, and G. HasÃ§elik, "A disposable mems dna biosensor for antibiotic resistant gene detection in staphylococcus aureus," in *Biomedical Engineering Meeting (BIYOMUT), 2010 15th National*, pp. 1–4, April 2010.

[136] V. Goyal, "Motion MEMS and its emerging automotive applications," *Auto Tech Review*, vol. 2, no. 7, pp. 54–55, 2013.

[137] S. O. Emmanuel and O. D. Peter, "Impact assessment of MEMS application on automobile driveability and functionality," in *Proceedings of the World Congress on Engineering*, vol. 3, 2013.

[138] J. Xie, C. Lee, and H. Feng, "Design, fabrication, and characterization of CMOS MEMS-based thermoelectric power generators," *Journal of Microelectromechanical Systems*, vol. 19, pp. 317–324, April 2010.

[139] C.-Y. Sue and N.-C. Tsai, "Human powered MEMS-based energy harvest devices," *Applied Energy*, vol. 93, pp. 390 – 403, 2012.

[140] A. Ros García, C. Montoliu Álvaro, V. Herrero Bosch, J. M. Monzó Ferrer, and R. J. Aliaga Varea, "Micro-generador termoeléctrico basado en contactos eléctricos pasantes," 2014. Spanish Patent ES2487590 A1.

[141] H. Hida, M. Shikida, K. Fukuzawa, S. Murakami, K. Sato, K. Asaumi, Y. Iriye, and K. Sato, "Fabrication of a quartz tuning-fork probe with a sharp tip for AFM systems," *Sensors and Actuators A: Physical*, vol. 148, no. 1, pp. 311 – 318, 2008.

[142] G. Schröpfer, M. de Labachelerie, S. Ballandras, and P. Blind, "Collective wet etching of a 3D monolithic silicon seismic mass system," *Journal of Micromechanics and Microengineering*, vol. 8, no. 2, p. 77, 1998.

[143] "Courtesy of sandia national laboratories, SUMMiT(TM) technologies." www.mems.sandia.gov.

[144] M. Tilli and A. Haapalinna, *Properties of silicon, in Silicon Based MEMS Materials & Technologies, Part 1, Chapter 1, Book Ed. V. Lindroos, M. Tilli A. Lehto and T. Motooka.* Micro & Nano Technologies, William Andrew/Elsevier, 2010.

[145] "Bosch leads Yole's MEMS market top 30 ranking.." http://www.analog-eetimes.com/en/bosch-leads-yole-s-mems-market-top-30-ranking.html?cmp_id=7&news_id=222906443.

[146] "Mems market overview: Steady growth for mems in 2013 and beyond.." http://www.semi.org/eu/sites/semi.org/files/docs/YOLE_MEMS%20Tech%20Seminar%202013-public.pdf.

[147] J.-j. Xiong, S.-j. Zheng, Y.-p. Hong, J. Li, Y.-l. Wang, W. Wang, and Q.-l. Tan, "Measurement of wireless pressure sensors fabricated in high temperature co-fired ceramic mems technology," *Journal of Zhejiang University SCIENCE C*, vol. 14, no. 4, pp. 258–263, 2013.

[148] S. Patil, V. Chu, and J. Conde, "Performance of thin film silicon MEMS on flexible plastic substrates," *Sensors and Actuators A: Physical*, vol. 144, no. 1, pp. 201–206, 2008.

[149] J. K. Luo, Y. Q. Fu, H. R. Le, J. A. Williams, S. M. Spearing, and W. I. Milne, "Diamond and diamond-like carbon MEMS," *Journal of Micromechanics and Microengineering*, vol. 17, no. 7, p. S147, 2007.

[150] D. Hwang, T. Saito, and N. Fujimori, “New etching process for device fabrication using diamond,” *Diamond and Related Materials*, vol. 13, pp. 2207–2210, 2004. Proceedings of the 9th International Conference on New Diamond Science and Technology (ICNDST-9).

[151] R. P. Feynman, “There’s plenty of room at the bottom,” *Engineering and science*, vol. 23, no. 5, pp. 22–36, 1960.

[152] J. Koo, M. Kim, and B. Kang, “Optical image stabilizer for camera lens assembly,” 2009. US Patent 7,489,340.

[153] W. Ma, H.-Y. Chan, C. C. Wong, C. Yiu, Y. C. Chan, and F. Lee, “Towards high resolution pico-projector applications: Design improvements on mems scanning mirror,” in *Nano/Micro Engineered and Molecular Systems (NEMS), 2011 IEEE International Conference on*, pp. 831–834, Feb 2011.

[154] M. Freeman, M. Champion, and S. Madhavan, “Scanned laser pico-projectors: Seeing the big picture (with a small device),” *Opt. Photon. News*, vol. 20, pp. 28–34, May 2009.

[155] “Challenged by newcomers proposing low-cost solutions, established MEMS companies must develop new strategies..” http://www.yole.fr/iso_upload/News/2014/PR_MEMS_Market_YOLEDEVELOPPEMENT_July2014.pdf.

[156] “Mems market to top $22 billion by 2018..” http://www.eetimes.com/document.asp?doc_id=1320035.

[157] F. Laermer and A. Schilp, “Plasma polymerizing temporary etch stop,” march 1996. US Patent 5501893.

[158] J. Parasuraman, A. Summanwar, F. Marty, P. Basset, D. E. Angelescu, and T. Bourouina, “Deep reactive ion etching of sub-micrometer trenches with ultra high aspect ratio,” *Microelectronic Engineering*, vol. 113, no. 0, pp. 35 – 39, 2014.

[159] M. J. Wolf, T. Dretschkow, B. Wunderle, N. Jurgensen, G. Engelmann, O. Ehrmann, A. Uhlig, B. Michel, and H. Reichl, “High aspect ratio tsv copper filling with different seed layers,” in *Electronic Components and Technology Conference*, pp. 563–570, 2008.

[160] M. Rimskog, “Through wafer via technology for MEMS and 3D integration,” in *Electronic Manufacturing Technology Symposium, 2007. IEMT ’07. 32nd IEEE/CPMT International*, pp. 286–289, Oct 2007.

[161] Y. Mita, M. Sugiyama, M. Kubota, F. Marty, T. Bourouina, and T. Shibata, “Aspect ratio dependent scalloping attenuation in DRIE and an application to low-loss fiber-optical switches,” in *Micro Electro Mechanical Systems,*

2006. MEMS 2006 Istanbul. 19th IEEE International Conference on, pp. 114–117, 2006.

[162] Ü. Sökmen, A. Stranz, S. Fündling, S. Merzsch, R. Neumann, H.-H. Wehmann, E. Peiner, and A. Waag, "Shallow and deep dry etching of silicon using ICP cryogenic reactive ion etching process," *Microsystem Technologies*, vol. 16, no. 5, pp. 863–870, 2010.

[163] J. Schille, *Investigation of micromachining using a high repetition rate femtosecond fibre laser*. PhD thesis, The University of Manchester, 2013.

[164] L. Xu and Y. Pan, "Electrochemical micromachining using vibrating tool electrode," *The International Journal of Advanced Manufacturing Technology*, vol. 75, no. 5-8, pp. 645–650, 2014.

[165] L.-M. Jiang, Y.-J. Du, J. Jia, L.-J. Lai, H. Zhou, L.-M. Zhu, Z.-W. Tian, Z.-Q. Tian, and D. Zhan, "Three dimensional micromachining on aluminum surface by electrochemical wet stamping technique," *Electrochemistry Communications*, vol. 33, no. 0, pp. 119–122, 2013.

[166] A. V. Singh, S. Chandra, S. Kumar, and G. Bose, "Mechanical and structural properties of RF magnetron sputter-deposited silicon carbide films for MEMS applications," *Journal of Micromechanics and Microengineering*, vol. 22, no. 2, p. 025010, 2012.

[167] J. Albero, L. Nieradko, C. Gorecki, H. Ottevaere, V. Gomez, H. Thienpont, J. Pietarinen, B. Päivänranta, and N. Passilly, "Fabrication of spherical microlenses by a combination of isotropic wet etching of silicon and molding techniques," *Opt. Express*, vol. 17, pp. 6283–6292, Apr 2009.

[168] M. Elwenspoek and H. V. Jansen, *Silicon Micromachining*. Cambridge Studies in Semiconductor Physics and Microelectronic Engineering, Cambridge University Press, 2004.

[169] C. Hedlund, U. Lindberg, U. Bucht, and J. Soderkvist, "Anisotropic etching of Z-cut quartz," *Journal of Micromechanics and Microengineering*, vol. 3, no. 2, p. 65, 1993.

[170] D. Zhuang and J. Edgar, "Wet etching of GaN, AlN, and SiC: a review," *Materials Science and Engineering: R: Reports*, vol. 48, no. 1, pp. 1–46, 2005.

[171] D. B. Lee, "Anisotropic etching of silicon," *Journal of Applied Physics*, vol. 40, no. 11, pp. 4569–4574, 1969.

[172] E. Ammar, T. Rodgers, and T. Rodgers, "UMOS transistors on 100 silicon," *Electron Devices, IEEE Transactions on*, vol. 27, pp. 907–914, May 1980.

[173] K. Fukuzawa, S. Terada, M. Shikida, H. Amakawa, H. Zhang, and Y. Mitsuya, "Mechanical design and force calibration of dual-axis micromechanical probe for friction force microscopy," *Journal of Applied Physics*, vol. 101, no. 3, 2007.

[174] N. Wilke, M. L. Reed, and A. Morrissey, "The evolution from convex corner undercut towards microneedle formation: theory and experimental verification," *Journal of Micromechanics and Microengineering*, vol. 16, no. 4, p. 808, 2006.

[175] P. Pal, M. A. Gosalvez, and K. Sato, "Silicon micromachining based on surfactant-added tetramethyl ammonium hydroxide: Etching mechanism and advanced applications," *Japanese Journal of Applied Physics*, vol. 49, no. 5R, p. 056702, 2010.

[176] H. Seidel, L. Csepregi, A. Heuberger, and H. Baumgärtel, "Anisotropic etching of crystalline silicon in alkaline solutions: I . orientation dependence and behavior of passivation layers," *Journal of The Electrochemical Society*, vol. 137, no. 11, pp. 3612–3626, 1990.

[177] O. Tabata, R. Asahi, H. Funabashi, K. Shimaoka, and S. Sugiyama, "Anisotropic etching of silicon in TMAH solutions," *Sensors and Actuators A: Physical*, vol. 34, no. 1, pp. 51 – 57, 1992.

[178] M. A. Gosálvez, I. Zubel, and E. Viinikka, *Wet etching of silicon, Chap. Ed. H. Seidel in Silicon Based MEMS Materials & Technologies, Part 4, Chapter 24, Book Ed. V. Lindroos, M. Tilli A. Lehto and T. Motooka.* Micro & Nano Technologies, William Andrew/Elsevier, 2010.

[179] M. A. Gosálvez *et al.*, *Atomistic modelling of anisotropic etching of crystalline silicon.* PhD thesis, Helsinki University of Technology, 2003.

[180] N. Ferrando, M. A. Gosálvez, and R. J. Colom, "Evolutionary continuous cellular automaton for the simulation of wet etching of quartz," *Journal of Micromechanics and Microengineering*, vol. 22, no. 2, p. 025021, 2012.

[181] P. Rangsten, C. Hedlund, I. V. Katardjiev, and Y. Bäcklund, "Etch rates of crystallographic planes in Z-cut quartz - experiments and simulation," *Journal of Micromechanics and Microengineering*, vol. 8, no. 1, p. 1, 1998.

[182] C. Hedlund, U. Lindberg, U. Bucht, and J. Soderkvist, "Anisotropic etching of z-cut quartz," *Journal of Micromechanics and Microengineering*, vol. 3, no. 2, p. 65, 1993.

[183] C. Kittel, *Introducción a la física del estado sólido.* Reverté, 1995.

[184] "Figure author: Felix Kling, licence CC BY 3.0 http://creativecommons.org/licenses/by/3.0/." Original link: http://en.wikipedia.org/wiki/Miller_index#/media/File:Miller_Indices_Felix_Kling.svg.

[185] D. Cheng, K. Sato, M. Shikida, A. Ono, K. Sato, K. Asaumi, and Y. Iriye, “Development of quartz etching database and 3-D micromachining simulation system,” in *Micromechatronics and Human Science, 2003. MHS 2003. Proceedings of 2003 International Symposium on*, pp. 281–285, Oct 2003.

[186] L. Levien, C. T. Prewitt, and D. J. Weidner, “Structure and elastic properties of quartz at pressure,” *American Mineralogist*, vol. 65, no. 9-10, pp. 920–930, 1980.

[187] N. Ferrando Jódar, *Estudio, Modelado e Implementación Paralela de Sistemas Celulares Utilizados en Microfabricación.* PhD thesis, Universitat Politècnica de València, 2011.

[188] K. Sato, M. Shikida, Y. Matsushima, T. Yamashiro, K. Asaumi, Y. Iriye, and M. Yamamoto, “Characterization of orientation-dependent etching properties of single-crystal silicon: effects of KOH concentration,” *Sensors and Actuators A: Physical*, vol. 64, no. 1, pp. 87–93, 1998. Tenth IEEE International Workshop on Micro Electro Mechanical Systems.

[189] K. Sato, M. Shikida, T. Yamashiro, K. Asaumi, Y. Iriye, and M. Yamamoto, “Anisotropic etching rates of single-crystal silicon for TMAH water solution as a function of crystallographic orientation,” *Sensors and Actuators A: Physical*, vol. 73, pp. 131–137, 1999.

[190] R. A. Wind, H. Jones, M. J. Little, and M. A. Hines, “Orientation-resolved chemical kinetics: Using microfabrication to unravel the complicated chemistry of KOH/Si etching,” *The Journal of Physical Chemistry B*, vol. 106, no. 7, pp. 1557–1569, 2002.

[191] M. A. Gosálvez, P. Pal, N. Ferrando, H. Hida, and K. Sato, “Experimental procurement of the complete 3D etch rate distribution of si in anisotropic etchants based on vertically micromachined wagon wheel samples,” *J. Micromech. Microeng.*, vol. 21, p. 125007, 2011.

[192] R. A. Wind and M. A. Hines, “Macroscopic etch anisotropies and microscopic reaction mechanisms: a micromachined structure for the rapid assay of etchant anisotropy,” *Surface Science*, vol. 460, pp. 21–38, 2000.

[193] I. Zubel and M. Kramkowska, “The effect of alcohol additives on etching characteristics in KOH solutions,” *Sensors and Actuators A: Physical*, vol. 101, no. 3, pp. 255–261, 2002.

[194] I. Zubel and M. Kramkowska, “Etch rates and morphology of silicon (h k l) surfaces etched in KOH and KOH saturated with isopropanol solutions,” *Sensors and Actuators A: Physical*, vol. 115, pp. 549–556, 2004. The 17th European Conference on Solid-State Transducers.

[195] P. Pal and K. Sato, "Various shapes of silicon freestanding microfluidic channels and microstructures in one-step lithography," *Journal of Micromechanics and Microengineering*, vol. 19, no. 5, p. 055003, 2009.

[196] D. Cheng, K. Sato, M. Shikida, A. Ono, K. Sato, K. Asaumi, and Y. Iriye, "Characterization of orientation-dependent etching properties of quartz: Application to 3-D micromachining simulation system," *Sensors and Materials*, vol. 17, no. 4, pp. 179–186, 2005.

[197] IntelliSense-Corp., "Intellietch webpage." http://www.intellisense.com/product.aspx?id=36, January 2015.

[198] C. H. Séquin, "Compute simulation of anisotropic crystal etching," *Solid-State Sensors and Actuators*, pp. 801–806, 1991.

[199] M. Zhao, H. Oigawa, J. Ji, and T. Ueda, "Application of a 2-D anisotropic etching simulator on perforated etching of quartz wafer," in *Sensors, 2011 IEEE*, pp. 1693–1696, Oct 2011.

[200] K. Asaumi, Y. Iriye, and K. Sato, "Anisotropic-etching process simulation system MICROCAD analyzing complete 3D etching profiles of single crystal silicon," *IEEE Micro Electro Mechanical Systems*, pp. 412–417, 1997.

[201] J. Fruhauf, K. Trautmann, J. Wittig, and D. Zielke, "A simulation tool for orientation dependent etching," *Journal of Micromechanics and Microengineering*, vol. 3, no. 3, p. 113, 1993.

[202] G. Li, T. Hubbard, and E. K. Antonsson, "SEGS: on-line WWW etch simulator," *MSM'98, Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuators*, 1998.

[203] M. A. Gosálvez, *Manufacture and Processing of MEMS Structures, in Silicon Based MEMS Materials & Technologies, Part 2, Chapter 10, Book Ed. V. Lindroos, M. Tilli A. Lehto and T. Motooka.* Micro & Nano Technologies, William Andrew/Elsevier, 2010.

[204] M. A. Gosálvez, "Visualtapas: an example of density functional theory assisted understanding and simulation of anisotropic etching (abstract only)," *Journal of Physics: Condensed Matter*, vol. 20, no. 6, p. 064234, 2008.

[205] A. J. Nijdam, E. van Veenendaal, H. M. Cuppen, J. van Suchtelen, M. L. Reed, J. G. E. Gardeniers, W. J. P. van Enckevort, E. Vlieg, and M. Elwenspoek, "Formation and stabilization of pyramidal etch hillocks on silicon 100 in anisotropic etchants: Experiments and monte carlo simulation," *Journal of Applied Physics*, vol. 89, no. 7, pp. 4113–4123, 2001.

[206] M. Gosálvez, R. Nieminen, P. Kilpinen, E. Haimi, and V. Lindroos, "Anisotropic wet chemical etching of crystalline silicon: atomistic monte-carlo simulations and experiments," *Applied Surface Science*, vol. 178, no. 1, pp. 7–26, 2001.

[207] Y. Xing, M. Gosálvez, and K. Sato, "Octree-search kinetic monte carlo algorithm for the simulation of complex 3D MEMS structures," in *Micro Electro Mechanical Systems, 2008. MEMS 2008. IEEE 21st International Conference on*, pp. 323–326, Jan 2008.

[208] M. A. Gosálvez, A. S. Foster, and R. M. Nieminen, "Multiscale modeling of anisotropic wet chemical etching of crystalline silicon," *EPL (Europhysics Letters)*, vol. 60, no. 3, p. 467, 2002.

[209] M. A. GosÃ¡lvez, D. Cheng, R. M. Nieminen, and K. Sato, "Apparent activation energy during surface evolution by step formation and flow," *New Journal of Physics*, vol. 8, no. 11, p. 269, 2006.

[210] Y. Xing, M. A. Gosálvez, K. Sato, M. Tian, and H. Yi, "Evolutionary determination of kinetic monte carlo rates for the simulation of evolving surfaces in anisotropic etching of silicon," *Journal of Micromechanics and Microengineering*, vol. 22, no. 8, p. 085020, 2012.

[211] N. Ferrando, M. A. Gosálvez, and A. Ayuela, "Evolutionary kinetic monte carlo: atomistic rates of surface-mediated processes from surface morphologies," *The Journal of Physical Chemistry C*, vol. 118, no. 22, pp. 11636–11648, 2014.

[212] O. Than and S. Büttgenbach, "Simulation of anisotropic chemical etching of crystalline silicon using a cellular automata model," *Sensors and Actuators A: Physical*, vol. 45, no. 1, pp. 85 – 89, 1994.

[213] H. Camon, A. Gue, J. Danel, and M. Djafari-Rouhani, "Modelling of anisotropic etching in silicon-based sensor application," *Sensors and Actuators A: Physical*, vol. 33, no. 1, pp. 103 – 105, 1992.

[214] T. J. Hubbard and E. K. Antonsson, "Cellular automata modeling in mems design," *Sensors and Materials*, vol. 9, pp. 437–448, 1997.

[215] S. Buttgenbach and O. Than, "Suzana: a 3d cad tool for anisotropically etched silicon microstructures," in *European Design and Test Conference, 1996. ED TC 96. Proceedings*, pp. 454–458, Mar 1996.

[216] Z. Zhu and C. Liu, "Micromachining process simulation using a continuous cellular automata method," *Microelectromechanical Systems, Journal of*, vol. 9, pp. 252–261, June 2000.

[217] Z. fa Zhou, Q. an Huang, W. hua Li, and W. Deng, "A cellular automaton-based simulator for silicon anisotropic etching processes considering high index planes," *Journal of Micromechanics and Microengineering*, vol. 17, no. 4, p. S38, 2007.

[218] M. A. Gosálvez, K. Sato, A. S. Foster, R. M. Nieminen, and H. Tanaka, "An atomistic introduction to anisotropic etching," *Journal of Micromechanics and Microengineering*, vol. 17, no. 4, p. S1, 2007.

[219] M. Gosálvez, Y. Xing, and K. Sato, "Analytical solution of the continuous cellular automaton for anisotropic etching," *Microelectromechanical Systems, Journal of*, vol. 17, pp. 410–431, April 2008.

[220] M. Gosálvez, Y. Xing, K. Sato, and R. Nieminen, "Discrete and continuous cellular automata for the simulation of propagating surfaces," *Sensors and Actuators A: Physical*, vol. 155, no. 1, pp. 98–112, 2009.

[221] P. Pal, M. Gosalvez, K. Sato, H. Hida, and Y. Xing, "Anisotropic etching on si{110}: experiment and simulation for the formation of microstructures with convex corners," *Journal of Micromechanics and Microengineering*, vol. 24, no. 12, p. 125001, 2014.

[222] M. A. Gosálvez, N. Ferrando, Y. Xing, P. Pal, K. Sato, J. Cerdá, and R. Gadea, "Simulating anisotropic etching of silicon in any etchant: evolutionary algorithm for the calibration of the continuous cellular automaton," *Journal of Micromechanics and Microengineering*, vol. 21, no. 6, p. 065017, 2011.

[223] B. Radjenović, M. Radmilović-Radjenović, and M. Mitrić, "Nonconvex hamiltonians in three dimensional level set simulations of the wet etching of silicon," *Applied Physics Letters*, vol. 89, no. 21, 2006.

[224] B. Radjenović and M. Radmilović-Radjenović, "3d simulations of the profile evolution during anisotropic wet etching of silicon," *Thin Solid Films*, vol. 517, no. 14, pp. 4233–4237, 2009. The proceedings of the 1st International Conference on Microelectronics and Plasma Technology (ICMAP 2008).

[225] B. Radjenović, M. Radmilović-Radjenović, and M. Mitrić, "Level set approach to anisotropic wet etching of silicon," *Sensors*, vol. 10, no. 5, pp. 4950–4967, 2010.

[226] M. Smiljanić, B. Radjenović, M. Radmilović-Radjenović, Ž. Lazić, and V. Jović, "Simulation and experimental study of maskless convex corner compensation in TMAH water solution," *Journal of Micromechanics and Microengineering*, vol. 24, no. 11, p. 115003, 2014.

[227] S. Franssila, *Front Matter.* Wiley Online Library, 2004.

[228] G. Oehrlein, “Reactive ion etching: Handbook of plasma processing technology,” *Park Ridge New Jersey*, vol. 196, 1990.

[229] J. P. Chang and J. W. Coburn, “Plasma-surface interactions,” *Journal of Vacuum Science & Technology A*, vol. 21, no. 5, pp. S145–S151, 2003.

[230] G. S. Oehrlein, J. F. Rembetski, and E. H. Payne, “Study of sidewall passivation and microscopic silicon roughness phenomena in chlorine-based reactive ion etching of silicon trenches,” *Journal of Vacuum Science & Technology B*, vol. 8, no. 6, pp. 1199–1211, 1990.

[231] G. S. Oehrlein and Y. Kurogi, “Sidewall surface chemistry in directional etching processes,” *Materials Science and Engineering: R: Reports*, vol. 24, no. 4, pp. 153–183, 1998.

[232] F. Laermer, S. Franssila, L. Sainiemi, and K. Kolari, *Deep Reactive Ion Etching, in Silicon Based MEMS Materials & Technologies, Part 4, Chapter 23, Book Ed. V. Lindroos, M. Tilli A. Lehto and T. Motooka.* Micro & Nano Technologies, William Andrew/Elsevier, 2010.

[233] W. C. Tian, J. W. Weigold, and S. W. Pang, “Comparison of Cl2 and F-based dry etching for high aspect ratio Si microstructures etched with an inductively coupled plasma source,” *Journal of Vacuum Science & Technology B*, vol. 18, no. 4, pp. 1890–1896, 2000.

[234] H. Lee and S. Wood, “Optimization of reactive ion etching (RIE) parameters for selective removal of MOSFET gate dielectric and evaluation of its physical and electrical properties,” *2012 NCUR*, 2012.

[235] B. Kastenmeier, P. Matsuo, J. Beulens, and G. Oehrlein, “Chemical dry etching of silicon nitride and silicon dioxide using CF4/O2/N2 gas mixtures,” *Journal of Vacuum Science & Technology A*, vol. 14, no. 5, pp. 2802–2813, 1996.

[236] H.-H. Hu, H.-Y. Lin, W. Fang, and B. C. Chou, “The diagnostic micromachined beams on (1 1 1) substrate,” *Sensors and Actuators A: Physical*, vol. 93, no. 3, pp. 258–265, 2001.

[237] Y. Wang, J. A. Henry, A. T. Zehnder, and M. A. Hines, “Surface chemical control of mechanical energy losses in micromachined silicon structures,” *The Journal of Physical Chemistry B*, vol. 107, no. 51, pp. 14270–14277, 2003.

[238] M. Boufnichel, P. Lefaucheux, S. Aachboun, R. Dussart, and P. Ranson, “Origin, control and elimination of undercut in silicon deep plasma etching in the cryogenic process,” *Microelectronic Engineering*, vol. 77, no. 3, pp. 327–336, 2005.

[239] R. J. Hoekstra, M. J. Kushner, V. Sukharev, and P. Schoenborn, "Microtrenching resulting from specular reflection during chlorine etching of silicon," *Journal of Vacuum Science & Technology B*, vol. 16, no. 4, pp. 2102–2104, 1998.

[240] A. P. Mahorowala and H. H. Sawin, "Etching of polysilicon in inductively coupled Cl2 and HBr discharges. II. simulation of profile evolution using cellular representation of feature composition and monte carlo computation of flux and surface kinetics," *Journal of Vacuum Science & Technology B*, vol. 20, no. 3, pp. 1064–1076, 2002.

[241] R. A. Gottscho, C. W. Jurgensen, and D. J. Vitkavage, "Microscopic uniformity in plasma etching," *Journal of Vacuum Science & Technology B*, vol. 10, no. 5, pp. 2133–2147, 1992.

[242] M. F. Doemling, N. R. Rueger, and G. S. Oehrlein, "Observation of inverse reactive ion etching lag for silicon dioxide etching in inductively coupled plasmas," *Applied Physics Letters*, vol. 68, no. 1, pp. 10–12, 1996.

[243] D. J. Economou, "Modeling and simulation of plasma etching reactors for microelectronics," *Thin Solid Films*, vol. 365, no. 2, pp. 348–367, 2000.

[244] R. J. Belen, S. Gomez, D. Cooperberg, M. Kiehlbauch, and E. S. Aydil, "Feature-scale model of si etching in SF6/O2 plasma and comparison with experiments," *Journal of Vacuum Science & Technology A*, vol. 23, no. 5, pp. 1430–1439, 2005.

[245] B. E. Thompson, H. H. Sawin, and D. A. Fisher, "Monte carlo simulation of ion transport through RF glow discharge sheaths," *Journal of Applied Physics*, vol. 63, no. 7, pp. 2241–2251, 1988.

[246] E. Baer, D. Kunder, J. Lorenz, M. Sekowski, and U. Paschen, "Coupling of monte carlo sputter simulation and feature-scale profile simulation and application to the simulation of back etching of an intermetal dielectric," in *Simulation of Semiconductor Processes and Devices (SISPAD), 2010 International Conference on*, pp. 53–56, September 2010.

[247] E. Baer, D. Kunder, P. Evanschitzky, and J. Lorenz, "Coupling of equipment simulation and feature-scale profile simulation for dry-etching of polysilicon gate lines," in *Simulation of Semiconductor Processes and Devices (SISPAD), 2010 International Conference on*, pp. 57–60, September 2010.

[248] E. Strasser and S. Selberherr, "Algorithms and models for cellular based topography simulation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, pp. 1104–1114, September 1995.

[249] E. Scheckler and A. R. Neureuther, "Models and algorithms for three-dimensional topography simulation with SAMPLE-3D," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, pp. 219–230, Feb 1994.

[250] R. J. Hoekstra, M. J. Grapperhaus, and M. J. Kushner, "Integrated plasma equipment model for polysilicon etch profiles in an inductively coupled plasma reactor with subwafer and superwafer topography," *Journal of Vacuum Science & Technology A*, vol. 15, no. 4, pp. 1913–1921, 1997.

[251] R. J. Hoekstra, M. J. Kushner, V. Sukharev, and P. Schoenborn, "Microtrenching resulting from specular reflection during chlorine etching of silicon," *Journal of Vacuum Science & Technology B*, vol. 16, no. 4, pp. 2102–2104, 1998.

[252] W. Guo, B. Bai, and H. H. Sawin, "Mixing-layer kinetics model for plasma etching and the cellular realization in three-dimensional profile simulator," *Journal of Vacuum Science & Technology A*, vol. 27, no. 2, pp. 388–403, 2009.

[253] J. A. Levinson, E. S. G. Shaqfeh, M. Balooch, and A. V. Hamza, "Ion-assisted etching and profile development of silicon in molecular and atomic chlorine," *Journal of Vacuum Science & Technology B*, vol. 18, no. 1, pp. 172–190, 2000.

[254] J. Lee, S. Yoon, and T. Won, "Topography simulation for structural analysis using cell advancing method," *Molecular Simulation*, vol. 31, no. 12, pp. 851–857, 2005.

[255] H. Yang, Y. Song, S. Zheng, L. Wang, and P. Jia, "An optimized-based ion etch yield modeling method in plasma etching," in *Control and Decision Conference (CCDC), 2013 25th Chinese*, pp. 2913–2918, May 2013.

[256] W. Guo and H. H. Sawin, "Review of profile and roughening simulation in microelectronics plasma etching," *Journal of Physics D: Applied Physics*, vol. 42, no. 19, p. 194014, 2009.

[257] G. Kokkoris, A. Tserepi, A. Boudouvis, and E. Gogolides, "Simulation of SiO2 and Si feature etching for microelectronics and microelectromechanical systems fabrication: A combined simulator coupling modules of surface etching, local flux calculation, and profile evolution," *Journal of Vacuum Science Technology A: Vacuum, Surfaces, and Films*, vol. 22, pp. 1896–1902, Jul 2004.

[258] B. Radjenović, J. K. Lee, and M. Radmilović-Radjenović, "Sparse field level set method for non-convex hamiltonians in 3D plasma etching profile simulations," *Computer Physics Communications*, vol. 174, no. 2, pp. 127–132, 2006.

[259] O. Ertl and S. Selberherr, “A fast level set framework for large three-dimensional topography simulations,” *Computer Physics Communications*, vol. 180, no. 8, pp. 1242 – 1250, 2009.

[260] O. Ertl and S. Selberherr, “Three-dimensional topography simulation using advanced level set and ray tracing methods,” in *Simulation of Semiconductor Processes and Devices, 2008. SISPAD 2008. International Conference on*, pp. 325–328, Sept 2008.

[261] “Anetch, physical etching simulator, release 0.7.5, Fraunhofer IISB, Erlangen,” 2009.

[262] R. Allegre, R. Chaine, and S. Akkouche, “A flexible framework for surface reconstruction from large point sets,” *Computer Graphics*, vol. 31, no. 2, pp. 190–204, 2007.

[263] J. C. Carr, R. K. Beatson, B. C. McCallum, W. R. Fright, T. J. McLennan, and T. J. Mitchell, “Smooth surface reconstruction from noisy range data,” in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE ’03, (New York, NY, USA), pp. 119–ff, ACM, 2003.

[264] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, “Piecewise smooth surface reconstruction,” in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH ’94, (New York, NY, USA), pp. 295–302, ACM, 1994.

[265] R. Mencl and H. Muller, “Graph-based surface reconstruction using structures in scattered point sets,” in *Computer Graphics International. Proceedings*, pp. 298–311, jun 1998.

[266] C. Oblonsek and N. Guid, “A fast surface-based procedure for object reconstruction from 3D scattered points,” *Computer Vision and Image Understanding*, vol. 69, no. 2, pp. 185–195, 1998.

[267] N. Amenta, M. Bern, and D. Eppstein, “The crust and the beta-skeleton: Combinatorial curve reconstruction,” in *Graphical Models and Image Processing*, pp. 125–135, 1998.

[268] N. Amenta, M. Bern, and M. Kamvysselis, “A new voronoi-based surface reconstruction algorithm,” in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH ’98, pp. 415–421, ACM, 1998.

[269] H. Edelsbrunner, “Shape reconstruction with delaunay complex,” in *Proceedings of the Third Latin American Symposium on Theoretical Informatics*, LATIN ’98, pp. 119–132, Springer-Verlag, 1998.

[270] T. K. Dey and S. Goswami, “Provable surface reconstruction from noisy samples,” in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG ’04, (New York, NY, USA), pp. 330–339, ACM, 2004.

[271] H.-K. Zhao, S. Osher, and R. Fedkiw, “Fast surface reconstruction using the level set method,” in *Variational and Level Set Methods in Computer Vision, 2001. Proceedings. IEEE Workshop on*, pp. 194–201, 2001.

[272] H.-K. Zhao, S. Osher, B. Merriman, and M. Kang, “Implicit and nonparametric shape reconstruction from unorganized data using a variational level set method,” *Computer Vision and Image Understanding*, vol. 80, no. 3, pp. 295–314, 2000.

[273] H. Liu, X. Wang, and W. Qiang, “Implicit surface reconstruction from 3D scattered points based on variational level set method,” in *Systems and Control in Aerospace and Astronautics, 2008. ISSCAA 2008. 2nd International Symposium on*, pp. 1–5, dec. 2008.

[274] H. Schröder and E. Obermeier, “A new model for si {100} convex corner undercutting in anisotropic {KOH} etching,” *Journal of Micromechanics and Microengineering*, vol. 10, no. 2, p. 163, 2000.

[275] M. Shikida, M. Ando, Y. Ishihara, T. Ando, K. Sato, and K. Asaumi, “Non-photolithographic pattern transfer for fabricating pen-shaped microneedle structures,” *Journal of Micromechanics and Microengineering*, vol. 14, no. 11, p. 1462, 2004.

[276] D. F. Weirauch, “Correlation of the anisotropic etching of singleâˆ’crystal silicon spheres and wafers,” *Journal of Applied Physics*, vol. 46, no. 4, pp. 1478–1483, 1975.

[277] D. Zielke and J. Frühauf, “Determination of rates for orientation-dependent etching,” *Sensors and Actuators A: Physical*, vol. 48, no. 2, pp. 151 – 156, 1995.

[278] M. Shikida, K. Sato, K. Tokoro, and D. Uchikawa, “Differences in anisotropic etching properties of KOH and TMAH solutions,” *Sensors and Actuators A: Physical*, vol. 80, no. 2, pp. 179 – 188, 2000.

[279] Z.-f. Zhou, Q.-a. Huang, and W.-h. Li, “Modeling and simulations of anisotropic etching of silicon in alkaline solutions with experimental verification,” *Journal of The Electrochemical Society*, vol. 156, no. 2, pp. F29–F37, 2009.

[280] M. Gosálvez, A. Foster, and R. Nieminen, “Atomistic simulations of surface coverage effects in anisotropic wet chemical etching of crystalline silicon,” *Applied Surface Science*, vol. 202, no. 3, pp. 160 – 182, 2002.

[281] J. A. Sethian and D. Adalsteinsson, “An overview of level set methods for etching, deposition, and lithography development,” *Semiconductor Manufacturing, IEEE Transactions on*, vol. 10, pp. 167–184, Feb 1997.

[282] B. Radjenović and M. Radmilović-Radjenović, “Level set simulations of the anisotropic wet etching process for device fabrication in nanotechnologies,” *Hemijska industrija*, vol. 64, no. 2, pp. 93–97, 2010.

[283] L. R. Arana, N. de Mas, R. Schmidt, A. J. Franz, M. A. Schmidt, and K. F. Jensen, “Isotropic etching of silicon in fluorine gas for mems micromachining,” *Journal of Micromechanics and Microengineering*, vol. 17, no. 2, p. 384, 2007.

[284] C. Liu, *Foundations of MEMS*. Illinois ECE series, Pearson Prentice Hall, 2006.

[285] A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker, “A streaming narrow-band algorithm: Interactive computation and visualization of level sets,” in *ACM SIGGRAPH 2005 Courses*, no. 243 in SIGGRAPH ’05, New York, NY, USA: ACM, 2005.

[286] A. Jalba, W. van der Laan, and J. Roerdink, “Fast sparse level sets on graphics hardware,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 19, pp. 30–44, Jan 2013.

[287] F. Galluzzo, N. Speciale, and O. Bernard, “A rigorous and efficient GPU implementation of level-set sparse field algorithm,” in *Image Processing (ICIP), 2012 19th IEEE International Conference on*, pp. 1705–1708, Sept 2012.

[288] “openMP web page.” http://openmp.org/wp/.

[289] “POSIX threads windows web page.” http://sourceware.org/pthreads-win32/.

[290] J. J. López, D. Carnicero, N. Ferrando, and J. Escolano, “Parallelization of the finite-difference time-domain method for room acoustics modelling based on CUDA,” *Mathematical and Computer Modelling*, vol. 57, no. 7, pp. 1822 – 1831, 2013.

[291] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP ’08, (New York, NY, USA), pp. 73–82, ACM, 2008.

[292] N. Ferrando, M. A. Gosálvez, J. Cerdá, R. Gadea, and K. Sato, "Faster and exact implementation of the continuous cellular automaton for anisotropic etching simulations," *Journal of Micromechanics and Microengineering*, vol. 21, no. 2, p. 025021, 2011.

[293] "Simode, collection of examples (c), gesselschaft fur mikroelektronikanwendung chemnitz mbh," 2001.

[294] C. hao Chang and H. Ming-Tsung, "A study of etch the anisotropic of quartz microstructures," Master's thesis, Department of Mechanical Engineering, National Central University, Taiwan, 2011.

[295] J. Liang, F. Kohsaka, T. Matsuo, and T. Ueda, "Wet etched high aspect ratio microstructures on quartz for MEMS applications," *IEEJ Transactions on Sensors and Micromachines*, vol. 127, no. 7, pp. 337–342, 2007.

[296] A. Zeniou, K. Ellinas, A. Olziersky, and E. Gogolides, "Ultra-high aspect ratio si nanowires fabricated with plasma etching: plasma processing, mechanical stability analysis against adhesion and capillary forces and oleophobicity," *Nanotechnology*, vol. 25, no. 3, p. 035302, 2014.

[297] H. Tsuda, Y. Takao, K. Eriguchi, and K. Ono, "Modeling and simulation of nanoscale surface rippling during plasma etching of si under oblique ion incidence," *Japanese Journal of Applied Physics*, vol. 51, no. 8S1, p. 08HC01, 2012.

[298] Y. Xiao, Z. Z. Fa, and L. W. Hua, "Three-dimensional simulation of profile evolution in plasma etching of polysilicon," in *Solid-State and Integrated Circuit Technology (ICSICT), 2014 12th IEEE International Conference on*, pp. 1–3, Oct 2014.

[299] L. Chiaramonte, R. Colombo, G. Fazio, G. Garozzo, and A. L. Magna, "A numerical method for the efficient atomistic simulation of the plasma-etch of nano-patterned structures," *Computational Materials Science*, vol. 54, no. 0, pp. 227 – 235, 2012.

[300] H. H. Hwang, T. R. Govindan, and M. Meyyappan, "Feature profile evolution simulation using a level set method," *Journal of The Electrochemical Society*, vol. 146, no. 5, pp. 1889–1894, 1999.

[301] Y. H. Im, Y. B. Hahn, and S. J. Pearton, "Level set approach to simulation of feature profile evolution in a high-density plasma-etching system," *Journal of Vacuum Science & Technology B*, vol. 19, no. 3, pp. 701–710, 2001.

[302] B. Radjenović and M. Radmilović-Radjenović, "3D etching profile evolution simulations: Time dependence analysis of the profile charging during SiO_2 etching in plasma," *Journal of Physics: Conference Series*, vol. 86, no. 1, p. 012017, 2007.

[303] O. Ertl, C. Heitzinger, and S. Selberherr, "Efficient coupling of monte carlo and level set methods for topography simulation," in *Simulation of Semiconductor Processes and Devices 2007* (T. Grasser and S. Selberherr, eds.), pp. 417–420, Springer Vienna, 2007.

[304] O. Ertl and S. Selberherr, "Three-dimensional plasma etching simulation using advanced ray tracing and level set techniques," *ECS Transactions*, vol. 23, no. 1, pp. 61–68, 2009.

[305] D. Zhang and M. J. Kushner, "Investigations of surface reactions during C_2F_6 plasma etching of SiO_2 with equipment and feature scale models," *Journal of Vacuum Science & Technology A*, vol. 19, no. 2, pp. 524–538, 2001.

[306] "Synopsys, inc." `http://www.synopsys.com`.

[307] "NVIDIA GeForce GTX titan GPU specifications." `http://www.nvidia.es/object/geforce-gtx-titan-es.html#pdpContent=2`.

[308] Q. Fang and D. Boas, "Tetrahedral mesh generation from volumetric binary and grayscale images," in *Biomedical Imaging: From Nano to Macro, 2009. ISBI '09. IEEE International Symposium on*, pp. 1142–1145, June 2009.

[309] F. Cacciola, "Triangulated surface mesh simplification," in *CGAL User and Reference Manual*, CGAL Editorial Board, 4.5 ed., 2014.