

Copyright © 1990, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**ALGORITHMS FOR THREE-DIMENSIONAL  
SIMULATION OF PHOTORESIST DEVELOPMENT**

by

Kenny Kai Hock Toh

Memorandum No. UCB/ERL M90/123

14 December 1990

**ALGORITHMS FOR THREE-DIMENSIONAL  
SIMULATION OF PHOTORESIST DEVELOPMENT**

by

Kenny Kai Hock Toh

Memorandum No. UCB/ERL M90/123

14 December 1990

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering  
University of California, Berkeley  
94720

**Algorithms for Three-Dimensional Simulation of Photoresist Development**  
**Copyright © 1990**  
**Kenny Kai Hock Toh**

**ALGORITHMS FOR THREE-DIMENSIONAL  
SIMULATION OF PHOTORESIST DEVELOPMENT**

by

Kenny Kai Hock Toh

Memorandum No. UCB/ERL M90/123

14 December 1990

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering  
University of California, Berkeley  
94720

TITLE PAGE

# ALGORITHMS FOR THREE-DIMENSIONAL SIMULATION OF PHOTORESIST DEVELOPMENT

Ph.D.

Kenny Kai Hock Toh

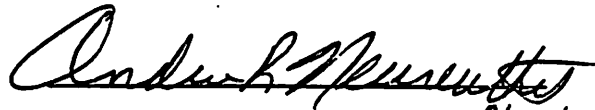
EECS Department

## ABSTRACT

A Three-Dimensional Optical Lithography Simulator has been developed based on a new ray-string algorithm for dissolution etch-front advancement. In developing the new algorithm, performance studies of cell, string and ray algorithms were carried out in two dimensions. A key finding was that a recursive ray method for the calculation of the surface-advancement vector produced numerically stable and highly accurate results. The optimum algorithm was found to be one that combines the recursive-ray method with the string approach, in which etch-rate-dependent rays are used to advance the nodes, segments and triangles which make up the etching boundary. This algorithm has been implemented in 3D in the C programming language, using a linked-list data structure to represent the etching boundary mesh. Recursive time-step selection, mesh modification, and clipping and delooping of the etch boundary surface have been implemented.

The 3D ray-string etch simulator has been coupled to 2D imaging and 3D resist-exposure simulators to form *SAMPLE-3D*, a complete fast and accurate 3D photolithography simulator. The complete simulator has been used to investigate the correlation between the

2D aerial image and the 3D developed resist profile. This includes applications to the printability of defects where the nonvertical resist dissolution effects play a strong role, as well as the design of phase-shifted masks where phase-transitions tend to print.



Prof. A. R. Neureuther

November 28, 1990

Committee Chairman

**Dedicated to my parents**



## ACKNOWLEDGEMENTS

I am deeply grateful to my advisor, Professor Andrew R. Neureuther for his invaluable guidance throughout the course of my graduate studies. I would also like to thank Professor William G. Oldham and Professor Leroy T. Kerth for serving on my dissertation committee. Discussions with them have further enhanced my technical understanding. I am also indebted to Professor Carlo Sequin, who served on my qualifying committee and also provided valuable technical insights.

It has been a great pleasure working with my colleagues in the IC processing and simulation groups at U.C. Berkeley. It is not possible to list all their names here. Nevertheless, I would especially like to thank Richard Ferguson, Nelson Tam, Dev Chen, Gino Addiego, Alex Wong, and Ed Scheckler for their friendship and for our many useful discussions. I will always remember our wild indoor football games in the hallowed corridors of Cory Hall. Thanks should also go to Giang Dao, Sid Das and Pat Troccoli of Intel Corporation, for their encouragement and support.

Last but not least, I thank my family for their love and support through the years. The sacrifice made on their part in order to provide me with an excellent education is greatly appreciated.

The financial support provided by SEMATECH and the Semiconductor Research Corporation is gratefully acknowledged.

## Table of Contents

Dedication .....	ii
Acknowledgements .....	iii
Table of Contents .....	iv
<b>CHAPTER 1 - INTRODUCTION .....</b>	<b>1</b>
1.1 : Simulation of Photolithography .....	1
1.2 : Etching Algorithms .....	3
1.3 : 3D Photolithography Simulators .....	4
1.4 : Research Outline .....	4
References for Chapter 1 .....	7
<b>CHAPTER 2 - PHOTOLITHOGRAPHY SIMULATION .....</b>	<b>10</b>
2.1 : 2D Photolithography Simulation with <i>SAMPLE</i> .....	10
2.2 : 3D Photolithography Simulation with <i>SPLAT</i> .....	15
2.2.1 : <i>SPLAT</i> : 2D Aerial Image Simulation .....	15
2.2.2 : 3D Simulation Using 2D Cutlines .....	18
2.3 : Complete 3D Photolithography Simulation .....	22
References for Chapter 2 .....	24
<b>CHAPTER 3 - THE CELL-REMOVAL ALGORITHM .....</b>	<b>27</b>
3.1 : Introduction .....	27
3.2 : The Algorithm for Cell-Removal .....	27

3.3 : Implementation of the Cell-Removal Algorithm .....	31
3.4 : Accuracy and the Cell-Removal Algorithm .....	31
3.4.1 : Faceting during Uniform Circular Etching .....	31
3.4.2 : Photoresist Etching .....	33
3.5 : Computation Time .....	37
3.6 : The Cell-Removal Algorithm in 3D .....	40
References for Chapter 3 .....	42
<b>CHAPTER 4 - THE MODIFIED CELL METHOD .....</b>	<b>44</b>
4.1 : Introduction .....	44
4.2 : The Huygens Principle Applied to Cell-Etching .....	44
4.3 : The Algorithm for Cell-Removal .....	45
4.4 : Implementation of the Modified Cell-Removal Algorithm .....	46
4.5 : Testing the Modified Cell-Removal Algorithm .....	48
4.5.1 : Uniform Circular Etching .....	48
4.5.2 : Etching with Triangular Analytic Functions .....	52
4.5.3 : Photoresist Etching .....	56
4.6 : Summary .....	59
References for Chapter 4 .....	61
<b>CHAPTER 5 - THE MATHEMATICAL BASIS OF THE SURFACE-</b>	
<b>ADVANCEMENT ALGORITHMS .....</b>	<b>62</b>
5.1 : Introduction .....	62
5.2 : Mathematical Formulation .....	63

5.2.1 : Formulation of the Problem .....	63
5.2.2 : The Etching Problem in One-Dimensional Space .....	65
5.2.3 : The Etching Problem in Three-Dimensional Space .....	68
5.2.4 : The Principle of Least Time .....	69
5.3 : Surface-Advancement Algorithms and the Principle of Least Time .....	72
References for Chapter 5 .....	76
<b>CHAPTER 6 - THE STRING AND THE RAY SURFACE-ADVANCEMENT</b>	
<b>ALGORITHMS .....</b>	<b>77</b>
6.1 : Introduction .....	77
6.2 : The String Algorithm .....	77
6.2.1 : The Algorithm for String Advancement .....	78
6.2.2 : Implementation of the String Algorithm .....	78
6.2.3 : Mesh Modification .....	80
6.2.3.1 : Rounding Errors at Diverging Corners .....	84
6.2.3.2 : Scissoring and Looping at Converging Corners .....	86
6.2.3.3 : Loop Formation in Photoresist Development .....	88
6.2.4 : The String Algorithm : Issues in 3D .....	92
6.3 : The Ray Algorithm .....	97
6.3.1 : The Algorithm for Advancing the Rays .....	98
6.3.2 : Implementation of the Ray Algorithm .....	99
6.3.3 : Testing the Ray Algorithm .....	102
6.3.3.1 : Etching with An Exponential Etch-Rate Function .....	102

6.3.3.2 : Ray-Spreading and Looping in Triangular Etch-Rates	104
6.3.3.3 : Ray-Spreading and Looping in Sinusoidal Etch-Rates	106
6.3.4 : The Ray Algorithm : Issues in 3D	113
6.4 : Summary : 3D Surface-Advancement Algorithms	114
References for Chapter 6	118
<b>CHAPTER 7 - THE RAY-STRING ALGORITHM : 2D IMPLEMENTATION</b>	
<b>ISSUES</b>	119
7.1 : Introduction	119
7.2 : Implementation of the Ray-String Algorithm in 2D	120
7.3 : The Algorithm for Surface Advancement	120
7.4 : Selection of the Size of the Time-Step	121
7.4.1 : Avoiding Abrupt Ray Reflection	121
7.4.2 : Recursive Vector Checking	125
7.4.3 : New Modifications to the Surface-Advancement Method	130
7.5 : Mesh Operations	131
7.5.1 : Mesh Modification	131
7.5.1.1 : Segment Cutting	132
7.5.1.2 : Segment Merging	134
7.5.2 : Mesh Clipping and Other Boundary Operations	139
7.5.2.1 : Mesh Clipping	139

7.5.2.2 : Boundary Flattening .....	143
7.5.3 : Mesh Delooping .....	146
7.5.3.1 : Loop Formation .....	146
7.5.3.2 : Loop Avoidance .....	148
7.5.3.3 : 2D Delooping .....	149
7.5.3.4 : Loop Formation in Photoresist Development .....	155
7.6 : Tuning Simulation Parameters for Accuracy .....	159
7.7 : Summary : 2D Implementation Issues .....	164
References for Chapter 7 .....	166
 <b>CHAPTER 8 - THE RAY-STRING ALGORITHM : 3D IMPLEMENTATION</b>	
<b>ISSUES .....</b>	<b>167</b>
8.1 : Introduction .....	167
8.2 : Implementation of the Ray-String Algorithm in 3D .....	168
8.2.1 : Data-Structure Requirements .....	168
8.2.2 : 3D Data-Structures .....	169
8.3 : The Algorithm for Surface-Advancement .....	171
8.4 : Mesh Operations .....	172
8.4.1 : Mesh Modification .....	172
8.4.1.1 : Segment Cutting .....	172
8.4.1.2 : Segment Merging .....	174
8.4.1.3 : Mesh Modification Procedures .....	184
8.4.2 : Mesh Clipping .....	185

8.4.3 : Mesh Delooping .....	187
8.5 : To Deloop or Not to Deloop .....	192
8.6 : 3D Simulation of Photoresist Development .....	194
8.7 : Computation Issues .....	197
8.8 : Summary : Implementation of a 3D Photoresist Development Simulator .....	200
References for Chapter 8 .....	201
<b>CHAPTER 9 - THREE-DIMENSIONAL SIMULATION OF OPTICAL LITHOG- RAPHY .....</b>	<b>202</b>
9.1 : Introduction .....	202
9.2 : Non-Vertical Resist Dissolution Effects .....	203
9.2.1 : The Intensity Threshold Model .....	203
9.2.2 : A Centered Opaque Defect in a Line-Space Array .....	207
9.2.3 : An Opaque Defect in a Corner .....	218
9.2.4 : Adjoining Phase-Shifted Spaces .....	232
9.3 : General Applications .....	234
9.3.1 : Clear-Field Lines & Spaces .....	238
9.3.2 : Fine Lines & Spaces on Negative Photoresist .....	238
9.3.3 : Projection Lens Aberrations .....	243
9.3.4 : Isolated Contacts with Optimally-Placed Phase-Shifters .....	246
9.4 : Summary .....	250
References for Chapter 9 .....	252

Chapter 10 - CONCLUSIONS AND FUTURE RESEARCH .....	255
10.1 : Conclusions .....	255
10.2 : Future Research .....	258
References for Chapter 10 .....	260
APPENDIX A - DERIVING THE DIFFERENTIAL RAY EQUATION .....	261
A.1 : Solving the Least Action Principle with Variational Calculus .....	262
A.2 : The Differential Ray Equation and The Eikonal .....	265
A.3 : Discretizing the Scalar Form of the Differential Ray Equation .....	268
A.3 : Discretizing the Vector Form of the Differential Ray Equation .....	269
References for Appendix A .....	273
APPENDIX B - USER GUIDES .....	274
<i>SAMPLE3D</i> User Guide .....	275
<i>PDRAW</i> User Guide .....	278
<i>CONTOUR</i> User Guide .....	282
<i>DRAWPLOT</i> User Guide .....	287
<i>DRAWMASK</i> User Guide .....	294
<i>CONV</i> User Guide .....	297



# CHAPTER 1

## INTRODUCTION

**li-thog-ra-phy** - the art or process of printing from a flat stone or metal plate: the design is put on the surface with a greasy material, and then water and printing ink are successively applied; the greasy parts, which repel water, absorb the ink, but the wet parts do not.

**pho-to-li-thog-ra-phy** - a printing process combining photography and lithography.

[From *Webster's New World Dictionary, 2nd edition, 1974*]

### 1.1. SIMULATION OF PHOTOLITHOGRAPHY

Photolithography simulation is increasingly being recognized as a critically important tool for the fabrication of modern integrated circuits. As integrated circuit device dimensions are pushed deeper and deeper into the submicron regime, the task of ensuring good yield and throughput with photolithography (also referred to as optical lithography) becomes more and more difficult. It is important to be able to understand and balance the many complex tradeoffs between materials, exposure tools, and wafer conditions that govern the pattern transfer process.

Simulation represents a powerful tool for studying photolithography, because it provides the means for systematically determining the effects of the many process parameters on the lithographic pattern transfer process. This systematic study can yield important information on the relationship between the various process parameters. It provides for a better understanding of the principles involved in the lithography process, which in turn allows for optimization of the performance of the pattern transfer process. And all this can be done much faster than experimental approaches, and at a fraction of the cost.

The simulation of optical lithography involves modeling the process by which patterns on a mask are transferred onto a photoresist-coated wafer via exposure to optical radiation. The origins of lithography process simulation can be traced to the 1970s. The essential ingredient was provided by Dill and co-workers,<sup>1, 2</sup> who formulated a quantitative model for the exposure and development of positive photoresists. In later years, this basic model was extended to multiple wavelengths.<sup>3</sup> A variety of optical effects such as defocus,<sup>4</sup> lens aberrations<sup>5</sup> and mask phase-shifting<sup>6</sup> were also added on. Dill's resist development model and the extensions to it have since been implemented in a number of photolithography simulation programs, including the *SAMPLE*<sup>7</sup> simulation program at U.C. Berkeley. † Since *SAMPLE* was introduced in April 1979, it has been used with great success for studying the issues involved in printing one-dimensional (1D) lines and spaces.

*SAMPLE* produces, principally, the two-dimensional cross-sections of line-edge profiles transferred from one-dimensional mask patterns. However, at small device dimensions, three-dimensional (3D) effects become more important in clearing out and properly filling corners, determining the effects of mask defects, and, in general, printing two-dimensional (2D) patterns such as elbows and squares. To deal with the issues related to patterning 2D mask patterns, it is necessary to be able to simulate the three-dimensional photoresist profile transferred from a two-dimensional mask pattern. And to do this, it is necessary to simulate the resist development, that is, the etching of the photoresist by an alkaline developer.

---

† *SAMPLE* is a FORTRAN program for Simulation And Modeling of Profiles in Lithography and Etching. It is capable of simulating the time evolution of topographical features of Integrated Circuit devices during multiple process steps. *SAMPLE* is being developed at the University of California at Berkeley by a student research group on process modeling and technology, under the guidance of Professors A.R. Neureuther and W.G. Oldham.<sup>8</sup>

## 1.2. ETCHING ALGORITHMS

The problem of etching photoresist can be generalized to that of modeling the shape or profile of a material as its surfaces are being etched. The etching is assumed to take place only on the surface of a time-varying front, and the etch-rate at each point in the volume of the material is given by some previously calculated etch-rate distribution. Furthermore, the photoresist is modeled as an inhomogeneous and isotropic medium.

In two dimensions, there are basically three algorithms for modeling etching. Dill's *cell* model,<sup>2</sup> introduced in 1975, was the very first algorithm to be used for development-etching simulation. The cell method is a volumetric algorithm; the material to be etched is divided into a matrix of tiny cells, and the etch surface is tracked by noting the etch state of each cell in the material. This algorithm, however, is slow and inefficient, and requires a frightening amount of computation time and memory in order to produce accurate results.

Jewett's *string* model<sup>9</sup> (used by *SAMPLE*) and Hagouel's *ray* model,<sup>10</sup> on the other hand, are faster and more accurate etching algorithms. These two models are both surface-advancement algorithms, in which a mesh of connected points is used to represent the surface of the material as it is being etched. However, the surface-advancement algorithms are difficult to implement, and also present difficult algorithmic and geometric problems in the treatment of boundaries and in loop formation.

These etching algorithms can still be used for modeling three-dimensional etching. But, as might be expected, the addition of an extra dimension does complicate matters considerably.

### 1.3. 3D PHOTOLITHOGRAPHY SIMULATORS

A number of 3D photolithography simulators have been introduced lately, the great majority of which use the cell method for 3D etching. The simulators written by Jones (*RD3D*),<sup>11</sup> Hirai,<sup>12</sup> and Bauer (*LITHSIM*)<sup>13</sup> all use the original cell algorithm. But in 3D, the cell method is painfully slow. The large memory required to run the simulation also limits the cell-based programs to supercomputers or large mainframes. There have been some recent attempts to correct the deficiencies of the cell method. The *SOLID* simulation program,<sup>14</sup> introduced recently by Pelka of the Fraunhofer Institute, and Mitsubishi's *3D-MULSS*<sup>15</sup> program both use faster modified versions of the cell method. Unfortunately, the accuracy of these simulators has not yet been established.

The string and ray algorithms are inherently faster and more accurate than the cell-based methods. However, these algorithms are quite difficult to implement in 3D. This difficulty is mirrored in the relatively small number of 3D simulators that use these algorithms. *RESPROT*<sup>16</sup> seems to be the only 3D process simulator using the string algorithm. Moniwa (*TRIPS-I*),<sup>17</sup> Jia<sup>18</sup> and Barouch<sup>19</sup> · <sup>20</sup> use variations of the ray algorithm for 3D etching. Despite its implementation difficulty, the 3D surface-advancement algorithm remains a most interesting area of research. There are many interesting tradeoffs, for example, speed vs. accuracy and front-smoothing vs. delooping, that have yet to be explored in detail.

### 1.4. RESEARCH OUTLINE

The work described in this document is aimed at implementing a fast, robust and accurate 3D lithography process simulator. In selecting an etching algorithm for 3D process simulation, it is especially important to consider the needs of process engineers. The process

simulator should be easy to use, and it should run on computing resources, such as workstations, that are easily accessible to process engineers. Therefore, the process simulator must be fast, and it should not need to use large amounts of computation memory. There should also be a good graphics interface for easy interpretation of the simulation results.

Special attention also needs to be paid to the accuracy and robustness of the etching simulation. Cell methods are stable, but slow and sometimes inaccurate. Even the ray and string algorithms produce inaccurate results if care is not taken during interpolation or boundary clipping. And in all three algorithms, there will be special cases which could disrupt the execution of the program. In the string algorithm, for example, the formation of loops could cause the program to crash.

Study of the literature reveals two clear areas of research in etching simulation. One safe and well-trod approach is to examine the cell method in greater detail, and to modify the algorithm to achieve greater speed and accuracy. The alternate approach is to delve into the intricacies of the surface-advancement algorithms. This latter method is high-risk, as it requires implementing the surface-advancement algorithms in 3D, which is no easy task. But at the same time, the inherent speed and accuracy advantages of the surface-advancement algorithms would provide a high payoff if these algorithms can be successfully implemented.

It was decided that a better choice of the etching algorithm for 3D simulation could be made if the etching algorithms themselves were better understood. Therefore, the first step in the decision process was to make a systematic and comprehensive comparison of the various etching algorithms in two dimensions. The algorithms for simulating the time-evolution of 2D etch profiles were implemented and compared in cost, convenience, and accuracy. In particular, a great deal of attention was paid to understanding and determining the conditions

under which the algorithms would provide accurate results.

Chapter 2 provides an overview of photolithography simulation in both two and three dimensions. The cell-removal algorithm for etching simulation is discussed in Chapter 3. One unexpected result of the study of the cell-removal algorithm was a promising new 3D cell-based algorithm, described in Chapter 4. Chapter 5 is devoted to a discussion of the mathematical basis of the surface-advancement algorithm. It is shown here that the string and the ray approaches are related closely to each other. The string and the ray algorithms are then examined in greater detail in Chapter 6, while an approach combining the two is discussed in Chapter 7.

Based on the study of 2D etching algorithms in Chapters 3-7, the algorithms that were more suited for 3D etching were selected and implemented. The ray-string algorithm, which combines the ray and string methods, showed the most potential for 3D etching simulation; its implementation in three dimensions is described in Chapter 8. The ray-string algorithm has been found to be both accurate and fast, and has been successfully coupled to 2D imaging and 3D resist-exposure simulators to form the basis of a complete 3D photolithography process simulator. In addition, supporting graphics programs have also been implemented and linked to the simulator for easy display of the simulation results. Application examples of the integrated 3D photolithography simulator are given in Chapter 9. Finally, the dissertation is concluded in Chapter 10.

## REFERENCES

1. F.H. Dill, W.P. Hornberger, P.S. Hauge, J.M. Shaw, "Characterization of Positive Photoresist," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 445-452, July 1975.
2. F.H. Dill, A.R. Neureuther, J.A. Tuttle, E.J. Walker, "Modeling Projection Printing of Positive Photoresists," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 456-464, July 1975.
3. M.A. Narasimham, F.H. Dill, *Projection Printer Photolithographic Images in Positive Photoresists Under Polychromatic Exposure*. Presented at the SPSE 30th Annual Conference, North Hollywood, CA, May 1977.
4. M.A. Narasimham, J.H. Carter, "Effects of Defocus on Photolithographic Images Made With Projection Printing Systems," *Proceedings of SPIE : Semiconductor Microlithography III*, 1978.
5. F.W. Wise, "Lens Aberrations and Nonuniform Illumination in Projection Lithography," *M.S. Thesis*, University of California, Berkeley, December 1981.
6. M.D. Prouty, A.R. Neureuther, "Optical Imaging with Phase-Shift Masks," *Proceedings of SPIE*, vol. 470, pp. 228-232, March 1984.
7. W. G. Oldham, S. N. Nandgaonkar, A. R. Neureuther, and M. M. O'Toole, "A General Simulator for VLSI Lithography and Etching Processes: Part I - Application to Projection Lithography," *IEEE Trans. on Electron Devices*, vol. ED-26, no. 4, pp. 717-722, April 1979. *SAMPLE* : 2D String method.
8. *SAMPLE 1.7a User Guide*, Electronics Research Laboratory, University of California, Berkeley, 1990.

9. R.E. Jewett, P.I. Hagouel, A.R. Neureuther, T. Van Duzer, "Line-Profile Resist Development Simulation Techniques," *Polymer Eng. Sci.*, vol. 17, no. 6, pp. 381-384, June 1977.
10. P.I. Hagouel, "X-ray Lithographic Fabrication of Blazed Diffraction Gratings," *Ph.D. Dissertation*, University of California, Berkeley, 1976.
11. F. Jones, J. Paraszczak, "RD3D (Computer Simulation of Resist Development in Three Dimensions)," *IEEE Transactions on Electron Devices*, vol. ED-28, no. 12, pp. 1544-1552, December 1981. *RD3D* : 3D Cell method.
12. Y. Hirai, M. Sasugo, M. Endo, K. Ikeda, S. Tomida, S. Hayama, "Three Dimensional Process Simulation for Photo and Electron Beam Lithography and Estimations of Proximity Effects," *Symposium on VLSI Technology*, p. 15, 1987. 3D Cell method.
13. J. Bauer, W. Mehr, U. Glaubitz, "Simulation and Experimental Results in 0.6  $\mu\text{m}$  Lithography using an I-Line Stepper," *Proceedings of SPIE : Optical/Laser Microlithography III*, vol. 1264, pp. 431-445, March 1990. *LITHSIM* : 3D Cell method.
14. J. Pelka, "SOLID : Comprehensive Three Dimensional Simulation Program for Optical Microlithography," *Information Brochure, Fraunhofer-Institut fur Mikrostrukturtechnik*, May 1990. *SOLID* : 3D Cell method.
15. M. Fujinaga, N. Kotani, T. Kunikiyo, H. Oda, M. Shirahata, Y. Akasaka, "Three-Dimensional Topography Simulation Model : Etching and Lithography," *IEEE Transactions on Electron Devices*, vol. ED-37, no. 10, pp. 2183-2192, October 1990. *3D-MULSS* : 3D Cell method.
16. T. Ito, K. Kadota, H. Fukui, M. Nagao, A. Sugimoto, M. Nozaki, T. Kato, "Submicrometer Pattern Correction for Optical Lithography," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 9-17, March 1988. *RESPROT* : 3D String method.



17. A. Moniwa, T. Matsuzawa, T. Ito, H. Sunami, "A Three-Dimensional Photoresist Imaging Process Simulator for Strong Standing-Wave Effect Environment," *IEEE Transactions on Computer Aided Design*, vol. CAD-6, no. 3, pp. 431-437, May 1987. *TRIPS-I*: 3D Ray-String method.
18. L. Jia, W. Jian-kun, W. Shao-jun, "Three-Dimensional Development of Electron Beam Exposed Resist Patterns Simulated by Using Ray Tracing Model," *Microelectronic Engineering*, vol. 6, pp. 147-151, 1987.
19. E. Barouch, B. Bradie, H. Fowler, S.V. Babu, "Three-Dimensional Modeling of Optical Lithography for Positive Photoresists," *Interface'89 : Proceedings of KTI Microelectronics Seminar*, pp. 123-136, November 1989. 3D Ray method.
20. E. Barouch, B. Bradie, S.V. Babu, "Calculation of Developed Resist Profiles by Least Action Principle," *Interface'88 : Proceedings of KTI Microelectronics Seminar*, pp. 187-196, November 1988. 2D Ray method.

## CHAPTER 2

### PHOTOLITHOGRAPHY SIMULATION

#### 2.1. 2D PHOTOLITHOGRAPHY SIMULATION WITH *SAMPLE*

The simulation of optical lithography involves modeling the process by which patterns on a mask are transferred onto a photoresist-coated wafer via exposure to optical radiation. Essentially, this process can be divided into three major components : imaging, exposure and development-etching. The *SAMPLE*<sup>1</sup> simulation program simulates the two-dimensional (2D) profile of the developed photoresist as a function of time by first calculating the aerial image intensity incident upon the photoresist (Figure 2.1). The exposure of the photoresist to light triggers chemical changes in the photoresist; the chemical changes (and modifications through baking and chemical amplification) are modeled using Dill's<sup>2</sup> algorithm (Figure 2.2). The exposure of the photoresist is related to the etch-rate in the volume of the photoresist by an empirically-constructed rate equation. This rate equation is used to generate a two-dimensional etch-rate distribution throughout the volume of the photoresist (Figure 2.3). This distribution is then used in a two-dimensional development-etch simulator to generate a two-dimensional profile of the photoresist (Figure 2.4).

Note in Figure 2.2 that the exposure of the photoresist is a strong function of distance into the photoresist. Because of multiple reflections between the surface of the photoresist and the substrate, standing waves are formed in the vertical intensity distribution. These standing waves affect the etch-rate distribution as well as the developed resist profile.

The approach outlined above to simulate the exposure of the photoresist avoids a time-consuming and rigorous solution of Maxwell's equations inside the photoresist by assuming

## Aerial Image Simulation

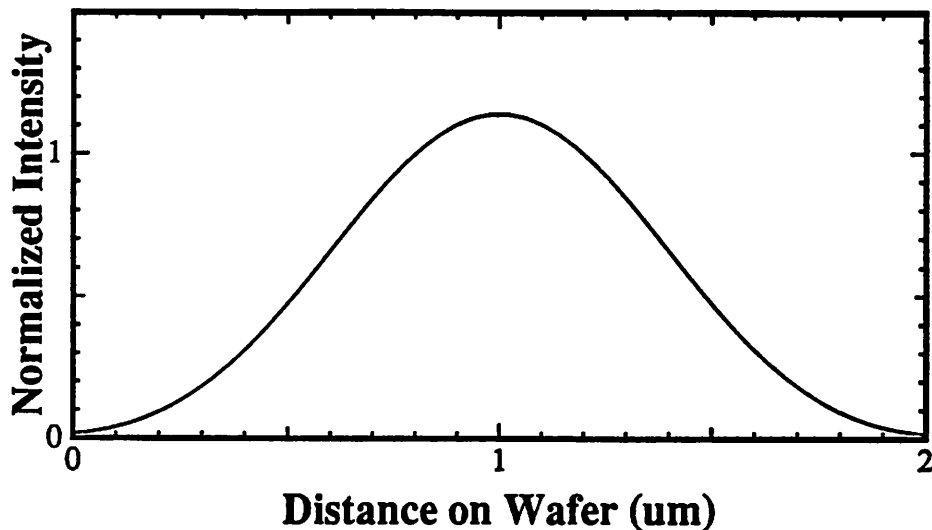


Figure 2.1 : Aerial image simulation of 1.25  $\mu\text{m}$  ( $0.8 \lambda/\text{NA}$ ) isolated space. The image was simulated using SPLAT, with  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA} = 0.28$ , and  $\sigma = 0.5$ .

## Normalized PAC Concentration, $M(x,z)$

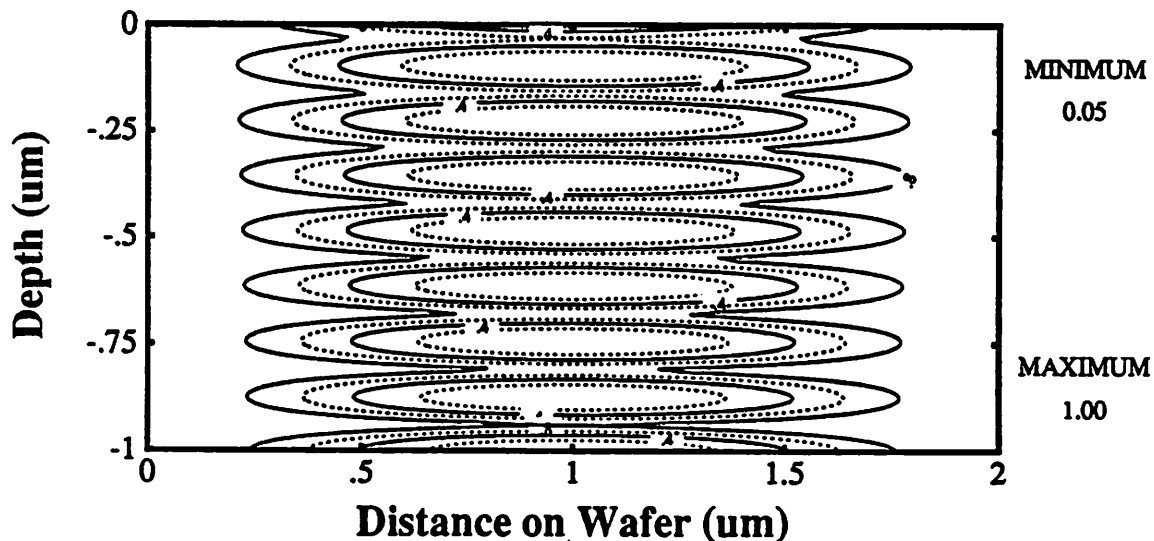


Figure 2.2 : Normalized concentration of photoactive compound, simulated from the aerial image of Figure 2.1. The simulation was performed on 1.0  $\mu\text{m}$  of Kodak 820 resist on 0.0741  $\mu\text{m}$  oxide. The resist exposure parameters are :  $A = 0.551 \mu\text{m}^{-1}$ ,  $B = 0.058 \mu\text{m}^{-1}$ ,  $C = 0.010 \text{ cm}^2/\text{mJ}$ .

## Etch Rate Distribution, $R(x,z)$

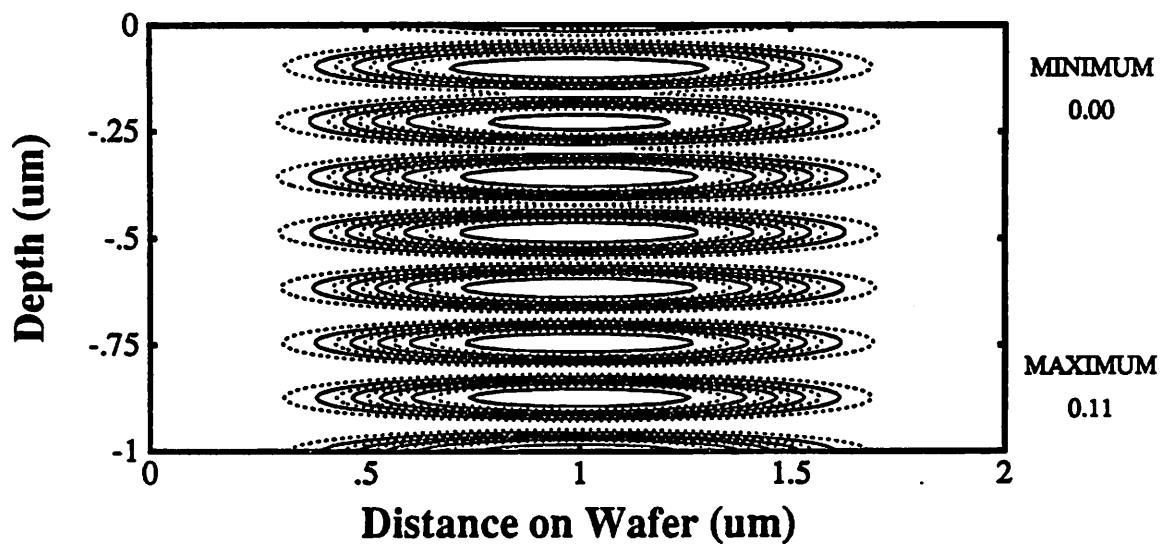


Figure 2.3 : Two-dimensional etch-rate distribution in the photoresist. The etch-rate is related to the photoactive compound concentration (Figure 2.2) by the Kim<sup>1</sup> rate-model, with parameters  $R1 = 0.1143 \mu\text{m/s}$ ,  $R2 = 0.0001683 \mu\text{m/s}$ ,  $R3 = 4.667$ ,  $R4 = 0.1 \mu\text{m}$ ,  $R5 = 0.45$ ,  $R6 = 0.3$ .

## SAMPLE String-Etch

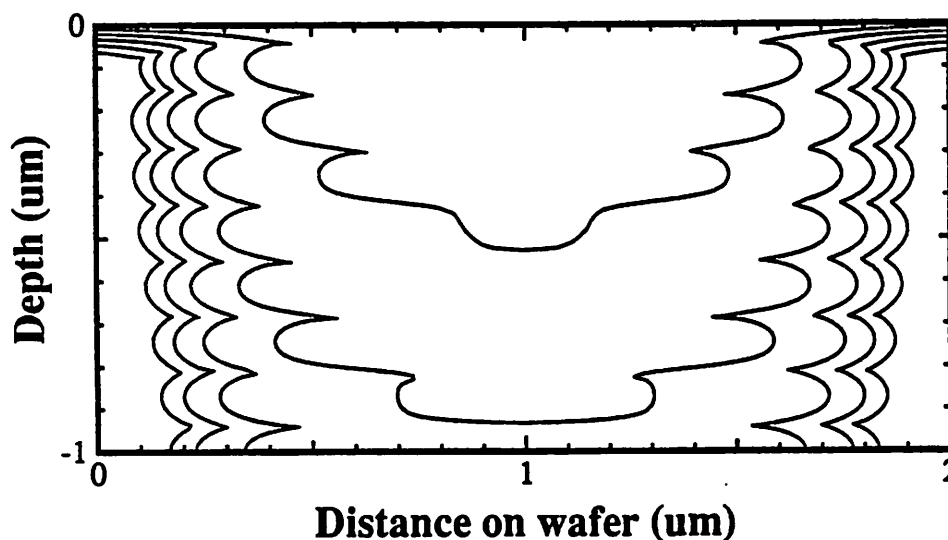


Figure 2.4 : Edge profiles for a  $1.25 \mu\text{m}$  isolated space in Kodak 820 resist developed for 15, 30, 45, 60 and 75 seconds. The profiles were simulated with *SAMPLE*, using the string algorithm.

<sup>1</sup> D.J. Kim, W.G. Oldham, A.R. Neureuther, "Development of Positive Photoresist," *IEEE Transactions on Electron Devices*, vol. ED-31, no. 12, pp. 1730-1735, December 1984.

that non-vertical ray propagation effects are negligible. The imaging and the exposure calculations then become separable, and the exposure calculation can then be performed using Berning's algorithm for reflection<sup>3</sup> assuming a single normally incident plane wave. Unfortunately, this assumption, while useful in the majority of simulations, is violated when large numerical aperture (NA) projection systems are used - the plane waves in such systems can be as much as 30° off-axis. For greater accuracy, it is necessary to apply electromagnetic diffraction theory to obtain a rigorous and comprehensive description of the imaging and exposure process in photolithography. A number of simulators that solve Maxwell's equations in two dimensions have been described recently in the literature.<sup>4 · 5 · 6</sup> Alternately, high numerical aperture projection printing may be simulated by including first order corrections to the vertical propagation model, as has been done by Bernard<sup>7</sup> and Mack.<sup>8</sup>

A flow diagram of the optical lithography simulation process is shown in Figure 2.5. As depicted in this diagram, there are two methods for simulating the exposure of the photoresist, both of which have been discussed. The rigorous method, in which Maxwell's equations are solved, treats the imaging and exposure as a single inseparable simulation step. This method, although more accurate than the vertical propagation model, is difficult to implement and is very computation-intensive. In contrast, the vertical propagation method, which allows the imaging and resist-exposure simulations to be separated, is much easier to implement and also less time-consuming. The resist development-etching simulation is independent of the exposure calculation; an empirically determined rate equation is used with either method to generate a etch-rate distribution in the resist. This distribution is then used to simulate the time-varying profile of the developing photoresist.

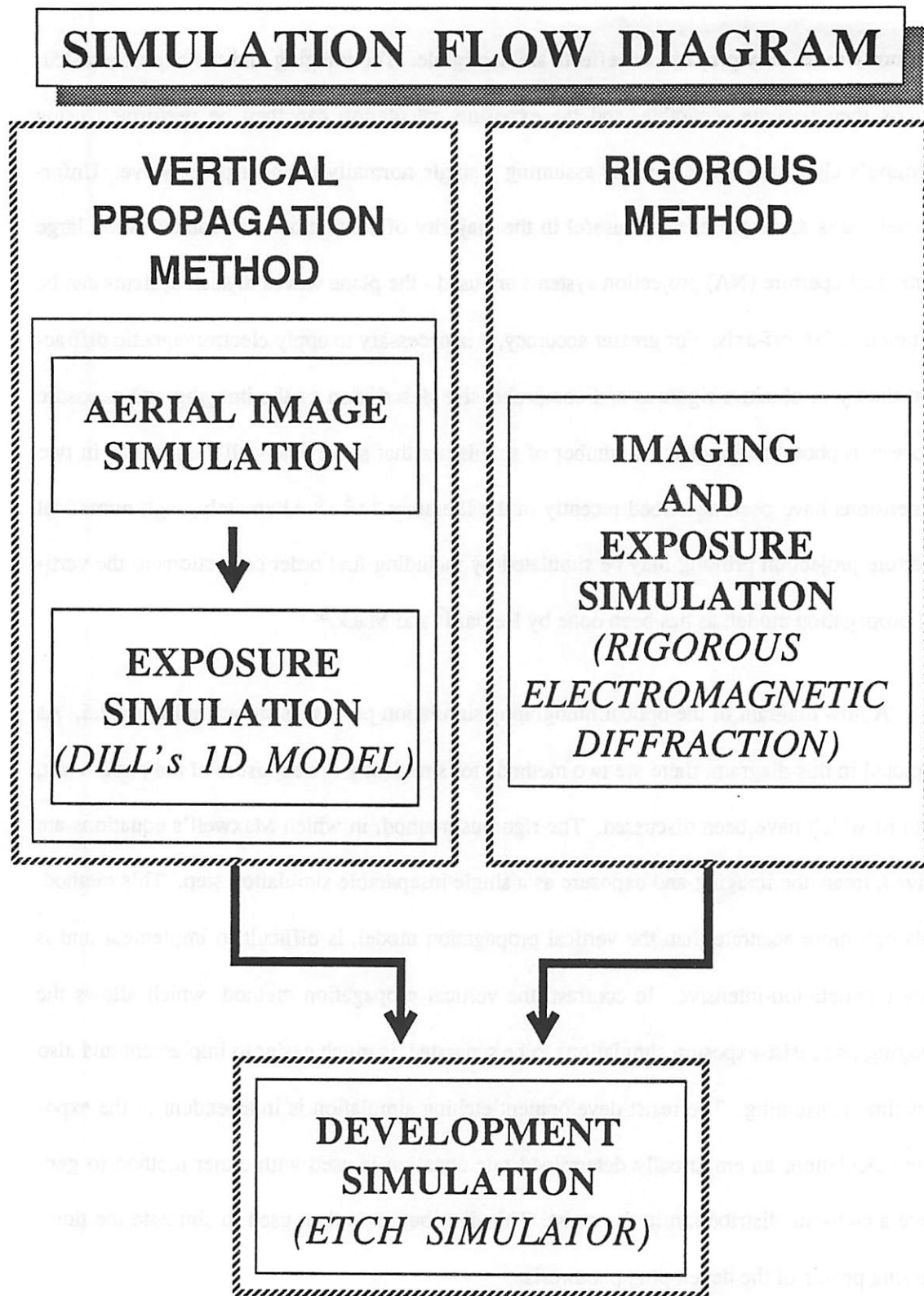


Figure 2.5 : Flow diagram for simulation of photoresist development.

## 2.2. 3D PHOTOLITHOGRAPHY SIMULATION WITH *SPLAT*

*SAMPLE* is a powerful tool for investigating the issues involved in printing one-dimensional (1D) lines and spaces. Unfortunately, in the real world, mask sets typically consist of two-dimensional patterns, such as elbows and squares. To properly analyze the effects of such two-dimensional mask patterns, a FORTRAN program, *SPLAT*,<sup>9</sup> was developed.

### 2.2.1. *SPLAT* : 2D Aerial Image Simulation

*SPLAT* (Simulation of Projection Lens Aberrations via TCCs) is a FORTRAN program, associated with *SAMPLE*, that simulates a two-dimensional optical image from a projection printer. The program calculates the intensity from an arbitrary two-dimensional mask pattern, using the Hopkins<sup>10</sup> theory of partially coherent imaging. The current version of the program can handle defocus effects, projection lens aberrations, as well as phase-shifted masks. *SPLAT* has been used to study feature-dependent effects in optical lithography, including proximity effects between neighboring features, clearing of corners and contact holes, and feature-dependent printing biases.<sup>11 · 12 · 13</sup> This simulation capability has also been used to design test patterns for isolating key optical parameters, and also for quantitatively interpreting the experimental results obtained using these test patterns.<sup>14 · 15 · 16</sup> And most recently, *SPLAT* has been used to systematically investigate the effects of phase-shifting masks in optical lithography.<sup>17 · 18 · 19</sup>

*SPLAT* by itself is not a complete tool, since it only simulates the image on the surface of the photoresist. It is possible to use *SPLAT* to generate intensity profiles along a cutline through a 2D mask, and then to feed that profile to *SAMPLE* for further resist development-etching. As an example, Figure 2.6 shows an image intensity contour plot of equal lines and

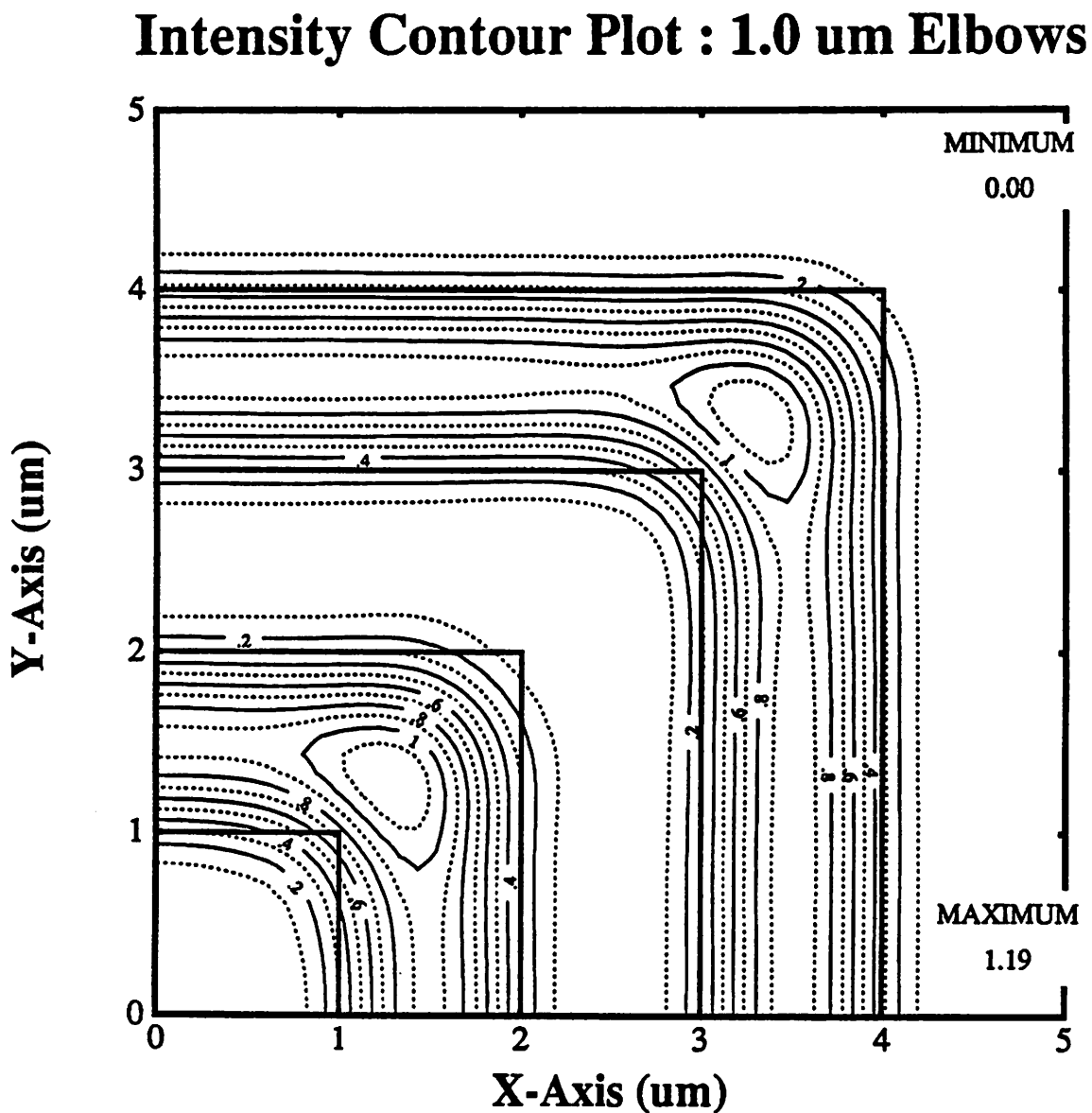
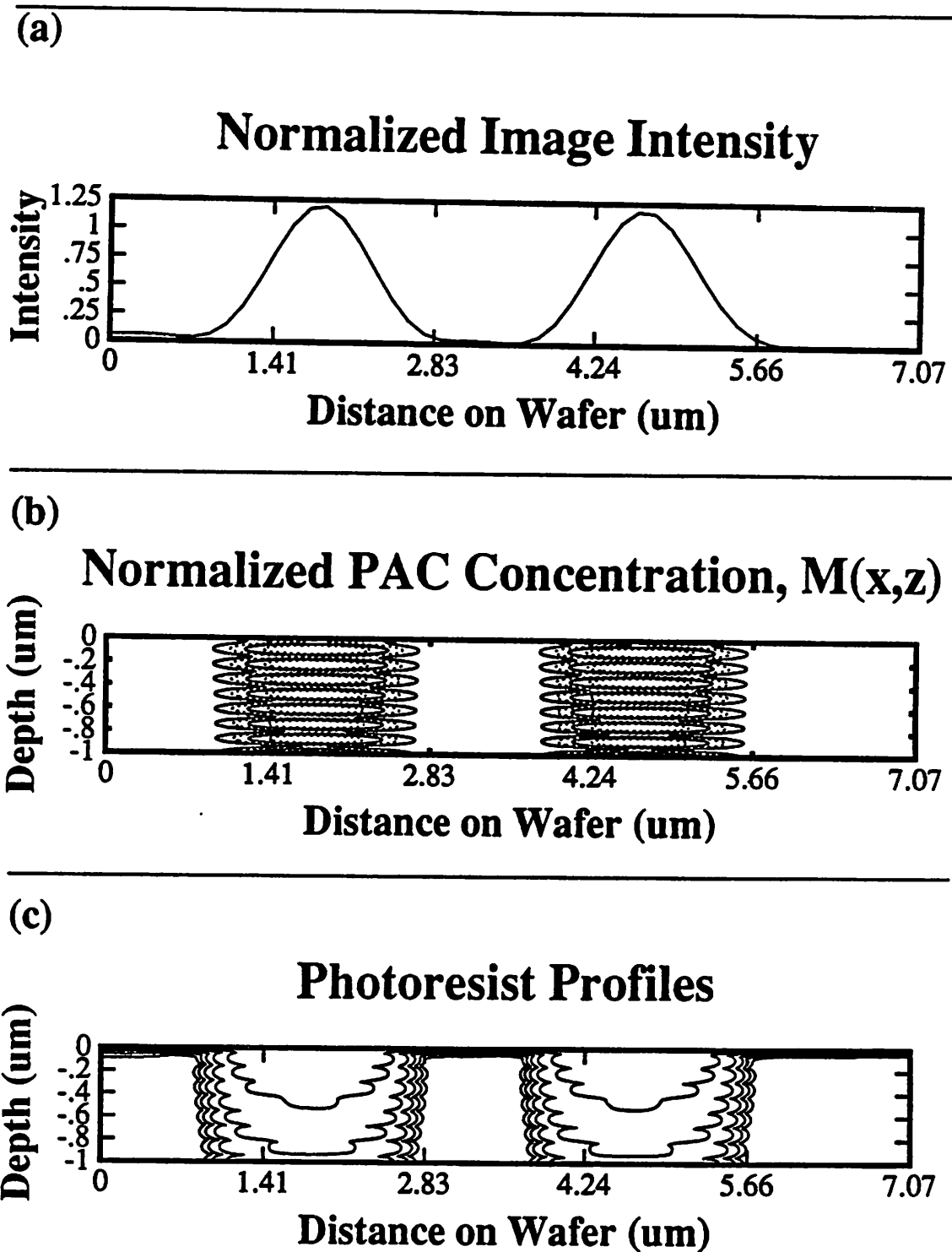


Figure 2.6 : *SPLAT* aerial image simulation of 1.0  $\mu\text{m}$  equal line-space elbows, run with wavelength  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA} = 0.28$ , and  $\sigma = 0.5$ . The mask outline is overlaid with dark solid lines. The image in Figure 2.7a is taken from a cutline along the diagonal, from (0.0,0.0) to (5.0,5.0).





**Figure 2.7 :** 2D lithography simulation along a cutline. The aerial image (a) is taken from a diagonal cutline of a *SPLAT*-simulated image of two elbows. (b) The PAC contours and (c) the resist profiles are simulated using *SAMPLE*.

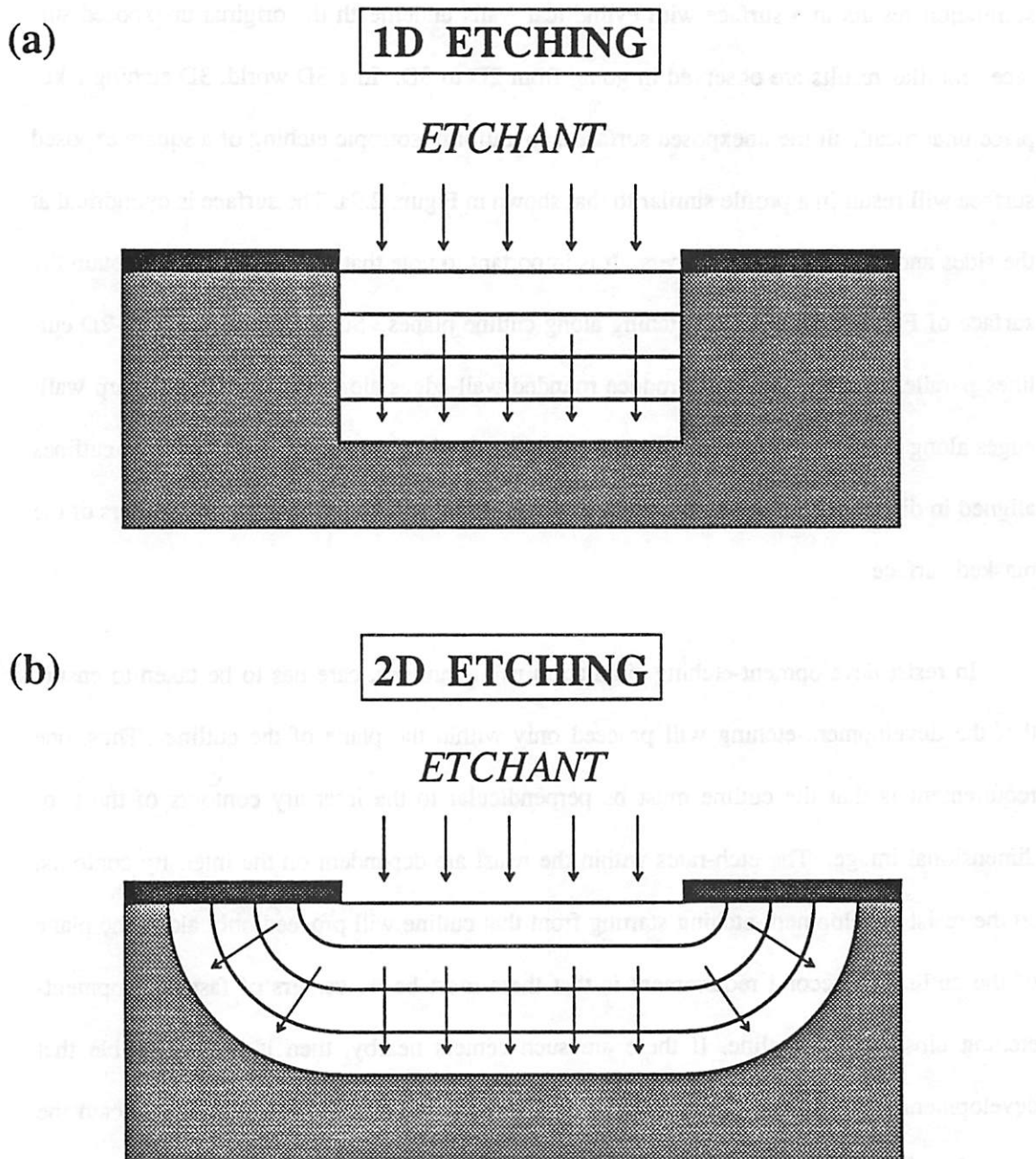
spaces in an elbow pattern. An intensity cutline along the corners of the elbows yields the intensity profile of Figure 2.7a. The plots in Figure 2.7b and 2.7c are produced when the *SPLAT*-generated intensity profile is fed into *SAMPLE* for resist development-etching.

### 2.2.2. 3D Simulation using 2D Cutlines

The 2D cutline procedure described previously is often used, not only in process simulation, to simulate three-dimensional (3D) behavior. Typically, a 3D volume is cut into parallel 2D planes, the simulation equations are solved along these planes, and the results are merged to form a "complete" solution. This pseudo-3D approach is usually used when the full 3D simulations are difficult to perform, or when simulators exist for 2D but not 3D calculations.

However, there are only certain conditions under which the cutline procedure will produce accurate results. This can be illustrated using the simple example of uniform isotropic etching, where a material exposed to an etchant is etched at a constant rate. In the examples shown in Figures 2.8 and 2.9, the material to be etched is exposed to an etchant along a masked surface, so that only a portion of the surface of the material is in contact with the etchant. As the material is etched away, the portions of the material underneath the unexposed surface will gradually come into contact with the etchant. Thus, etching proceeds uniformly from the initial exposed surface. In a 2D world, this uniform two-dimensional etching results in a surface similar to that shown in Figure 2.8b. In 1D, however, etching will proceed only along a one-dimensional line perpendicular to the original surface. A point on the exposed surface will move straight down into the material as it is being etched, while a point on the unexposed surface will not move at all. This result is shown in Figure 2.8a.

## THE NEED FOR 2D ETCHING



**Figure 2.8 :** (a) 1D uniform etching along an exposed surface. The material underneath the unexposed surface remains unetched.  
 (b) 2D uniform etching along an exposed surface. The material underneath the unexposed surface is etched uniformly.

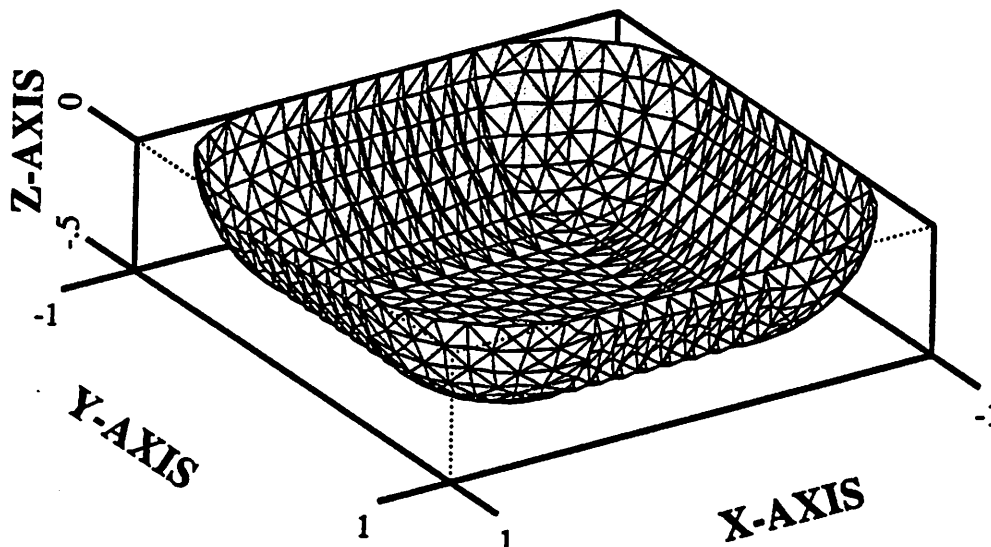
It is quite clear that there is a vast improvement in the etching simulation as one goes from 1D to 2D. The 1D simulation produces a box-like surface with  $90^\circ$  walls, while the 2D simulation results in a surface with cylindrical walls underneath the original unexposed surface. Similar results are observed in going from 2D to 3D. In a 3D world, 3D etching takes place underneath all the unexposed surfaces, so uniform isotropic etching of a square exposed surface will result in a profile similar to that shown in Figure 2.9a. The surface is cylindrical at the sides and spherical at the corners. It is important to note that it is impossible to obtain the surface of Figure 2.9a with 2D etching along cutline planes. Such a procedure with 2D cutlines parallel to the  $x$ -axis will produce rounded wall-edges along the  $x$ -axis and sharp wall-edges along the  $y$ -axis (Figure 2.9b). This could be fixed, of course, by using multiple cutlines aligned in different directions. But even so, there would still be problems at the corners of the masked surface.

In resist development-etching simulation using cutlines, care has to be taken to ensure that the development-etching will proceed only within the plane of the cutline. Thus, one requirement is that the cutline must be perpendicular to the intensity contours of the two-dimensional image. The etch-rates within the resist are dependent on the intensity contours, so the resist development-etching starting from that cutline will proceed only along the plane of the cutline. A second requirement is that there must be no centers of fast development-etching close to the cutline. If there are such centers nearby, then it is conceivable that development-etching from these centers might cross the plane of the cutline underneath the original resist surface. It is not easy to determine if this last requirement can be met, since there is a nonlinear relationship between the intensity incident on the resist and the actual etch-rates within the volume of the resist. In the example of Figures 2.6 and 2.7, the first criteria is met; the intensity contours are normal to the diagonal cutline, so etching from the ori-

# THE NEED FOR 3D ETCHING

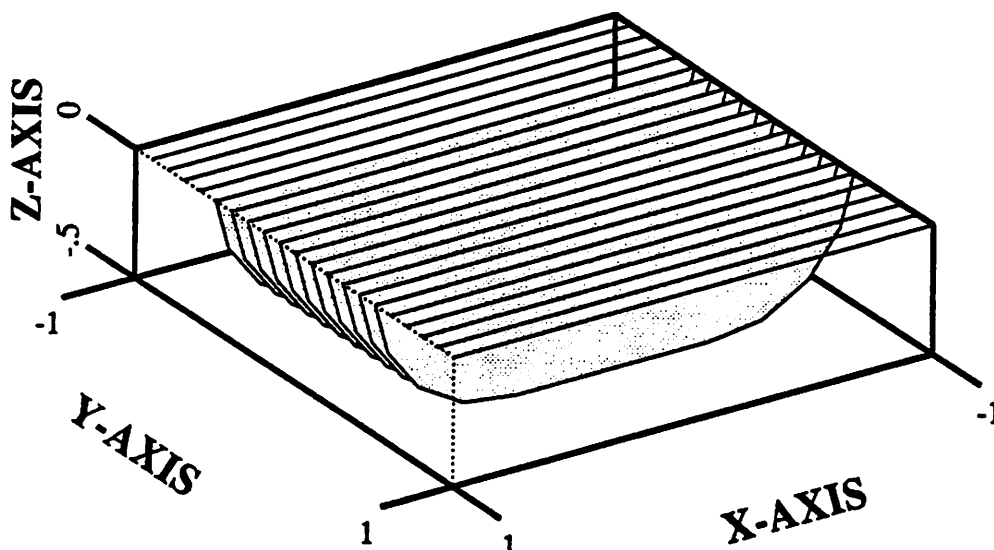
(a)

## 3D ETCHING



(b)

## 3D ETCHING USING 2D CUTLINES



**Figure 2.9:** (a) 3D uniform etching along an exposed surface. The material underneath the unexposed surface is etched uniformly.

(b) 3D etching using 2D cutlines. The composite result is not accurate, since etching does not take place underneath the edges parallel to the x-axis.

ginally flat surface will proceed only in the cutline plane. However, it is possible that the second criteria will not be met; the standing waves caused by multiple reflections between the resist and substrate might result in etching that crosses the diagonal cutline plane. These standing waves could affect the final resist profile.

Modeling 3D etching with 2D etching along cutlines is potentially fraught with the danger of not meeting the two requirements discussed above. This makes it very difficult to implement 3D etching using 2D multiple cutlines.

### 2.3. COMPLETE 3D PHOTOLITHOGRAPHY SIMULATION

In order to properly investigate issues associated with 2D mask patterns, it is much better to use a complete 3D photolithography simulator. The simulation process flow described in Figure 2.5 also applies for 3D simulation. Just as in 2D, the three-dimensional simulation of photolithography can be performed assuming vertical propagation. In this case, the simulation can be broken into the three major components described earlier - imaging, exposure and development-etching. Alternately, the imaging and exposure could be treated as a whole, and solved for rigorously. However, the rigorous electromagnetic diffraction method is computation-intensive even in 2D; a rigorous 3D solution of Maxwell's equations will undoubtedly be much more time-consuming.

Given the advantages of the vertical propagation scheme, it makes a great deal of sense to implement a 3D resist simulator with modular and separate imaging, exposure and development-etching parts. The first two components of this simulator are not difficult to build. *SPLAT* already exists, and extending Dill's 1D exposure model to 3D is straightforward. However, writing a 3D development-etch simulator is a more difficult proposition,

for a careful choice of algorithms and data structures has to be made in order to obtain a fast, robust and efficient etch simulator. But if the 3D development-etch simulator is developed and implemented carefully, it can be used with both vertical-propagation and rigorous 3D resist-exposure simulators for complete and accurate simulation of photolithography.

## REFERENCES

1. W. G. Oldham, S. N. Nandgaonkar, A. R. Neureuther, and M. M. O'Toole, "A General Simulator for VLSI Lithography and Etching Processes: Part I - Application to Projection Lithography," *IEEE Trans. on Electron Devices*, vol. ED-26, no. 4, pp. 717-722, April 1979. *SAMPLE* : 2D String method.
2. F.H. Dill, A.R. Neureuther, J.A. Tuttle, E.J. Walker, "Modeling Projection Printing of Positive Photoresists," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 456-464, July 1975.
3. P.H. Berning, "Theory and Calculations of Optical Thin Films," in *Physics of Thin Films*, ed. George Hass, vol. I, pp. 69-121, Academic Press, New York, 1963.
4. J. Gamelin, R. Guerrieri, A.R. Neureuther, "Exploration of Scattering from Topography with Massively Parallel Computers," *J. Vac. Sci. Technol. B.*, 1989.
5. M.S. Yeung, "Modeling High Numerical Aperture Optical Lithography," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 149-167, March 1988.
6. H.P. Urbach, D.A. Bernard, "Modeling Latent Image Formation in Photolithography using the Helmholtz Equation," *Proceedings of SPIE : Optical/Laser Microlithography III*, vol. 1264, pp. 278-293, March 1990.
7. D.A. Bernard, "Simulation of Focus Effects in Photolithography," *IEEE Transactions on Semiconductor Manufacturing*, vol. 1, no. 3, pp. 85-97, August 1988.
8. C.A. Mack, "Understanding Focus Effects in Submicron Optical Lithography," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 135-148, March 1988.



9. Kenny K.H. Toh, "Two-Dimensional Images with Effects of Lens Aberrations in Optical Lithography," *M.S. Thesis, Memorandum No. UCB/ERL M88/30*, University of California, Berkeley, May 20, 1988.
10. H.H. Hopkins, "On the Diffraction Theory of Optical Images," *Proc. Royal Soc. Series A.*, vol. 217, no. 1131, pp. 408-432, 1953.
11. B. Huynh, K.K.H. Toh, W.E. Haller, A.R. Neureuther, "Optical Printability of Defects in Two-Dimensional Patterns," *J. Vac. Sci. Technol. B 6(6)*, pp. 2207-2212, Nov/Dec 1988.
12. K.K.H. Toh, C.C. Fu, K.L. Zollinger, A.R. Neureuther, R.F.W. Pease, "Characterization of Voting Suppression of Optical Defects Through Simulation," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 194-202, March 1988.
13. V. Mastromarco, A.R. Neureuther, K.K.H. Toh, "Printability of Defects in Optical Lithography : Polarity and Critical Location Effects," *J. Vac. Sci. Technol. B 6(1)*, pp. 224-229, Jan/Feb 1988.
14. K.K.H. Toh and A.R. Neureuther, "Mapping Image Quality in Optical Lithography with Visual Test Patterns," *Interface'89 : Proceedings of KTI Microelectronics Seminar*, pp. 155-177, November 1989.
15. A.R. Neureuther, K.K.H. Toh, J.E. Fleishman, D. Yu, G. Misium, B. Huynh, B. Uathavikul, W.G. Oldham, "Exploratory Test Structures for Image Evaluation in Optical Projection Printing," *Proceedings of SPIE : Optical/Laser Microlithography II*, vol. 1088, pp. 83-91, March 1989.
16. K.K.H. Toh, A. R. Neureuther, "Identifying and Monitoring Effects of Lens Aberrations in Projection Printing," *Proceedings of SPIE : Optical Microlithography VI*, vol. 772, pp. 202-209, March 1987.

17. K.K.H. Toh, "Optical Lithography with Chromeless Phase-Shifted Masks," *Intel Internal Memorandum*, August 1990.
18. K.K.H. Toh, G. Dao, R. Singh, H. Gaw, *Chromeless Phase-Shifted Masks : A New Approach to Phase-Shifting Masks*. Presented at the BACUS 10th Annual Symposium on Microlithography, September 1990.
19. K.K.H. Toh, "Design Methodology for Dark-Field Phase-Shifted Masks," *Intel Internal Memorandum*, August 1990.

## CHAPTER 3

### THE CELL-REMOVAL ALGORITHM

#### 3.1. INTRODUCTION

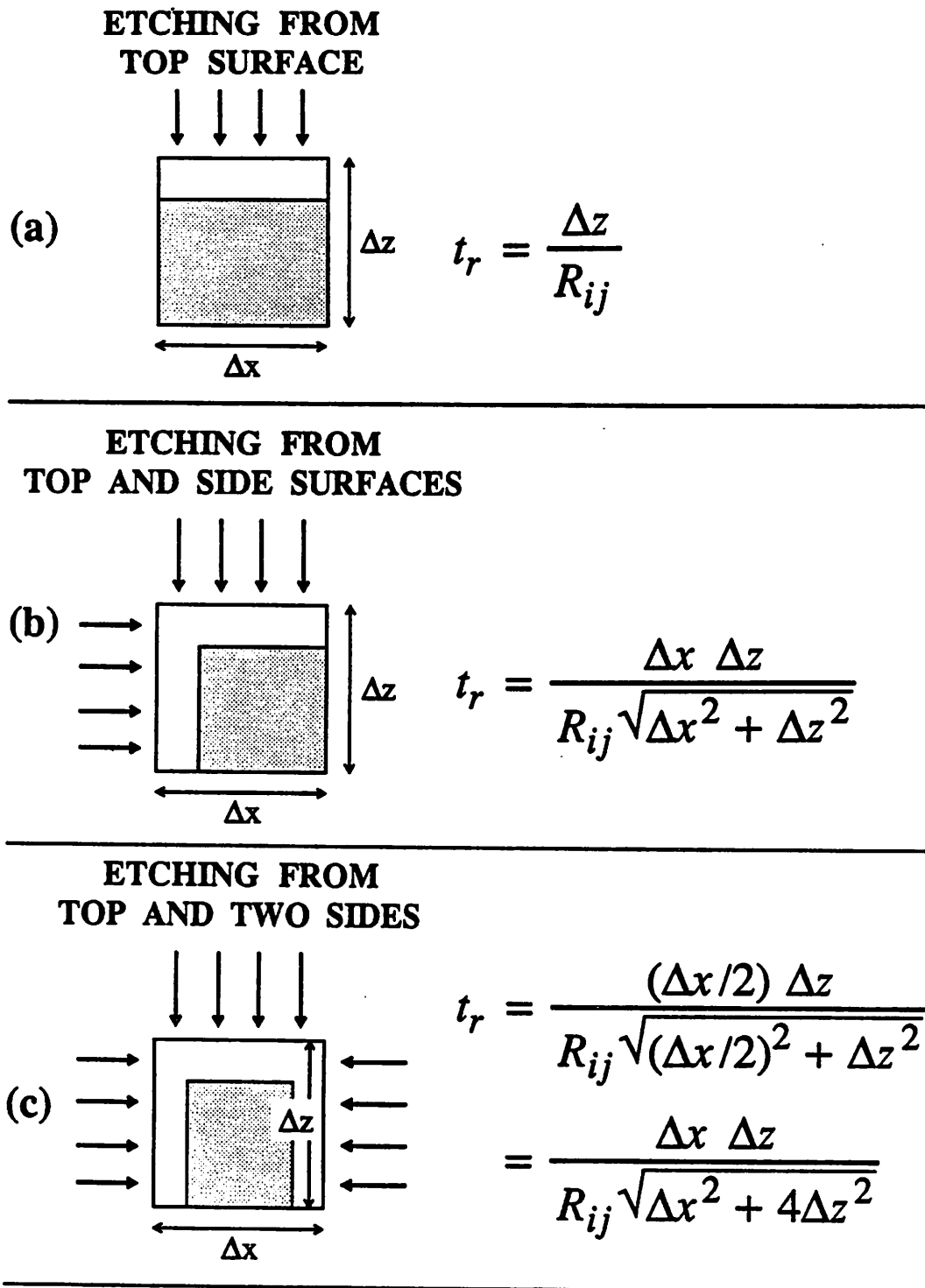
The cell-removal algorithm is a volume etching method that was first proposed for etching simulation by Dill, et. al.,<sup>1</sup> in 1975. It has remained quite popular, with both 2D and 3D implementations having been reported in the literature.<sup>2,3,4,5,6</sup>

The cell-removal method, as originally proposed by Dill, divides the the material being etched into rectangular cells, each characterized as completely etched, completely unetched, or partially etched. The surface or etching boundary consists of unetched or partially etched cells that are in contact with fully etched cells. The cells on the etching boundary are exposed to the etchant, and etching proceeds along this surface. During the etch process, cells are removed by the etchant according to the local etch-rate and the number of sides of the cell in contact with the etchant. When an old cell is removed, the new cells exposed are allowed to start etching.

#### 3.2. THE ALGORITHM FOR CELL-REMOVAL

The algorithm for moving the etch surface can be stated as follows :

- [I] For each cell on the etching boundary, find the etch-time  $t_r$  needed to remove the cell. The etch-time is calculated according to the etch-rate at the center of the cell, and the number of sides of the cell in contact with the etchant. The equations governing the etch-times for typical cases are shown in Figure 3.1.



**Figure 3.1 :** Time to remove cells for different exposure conditions. In the 3 cases above, a cell is exposed along (a) the top surface, (b) the top and a side surface, and (c) the top and two sides.

[II] Remove the cell that has etched the fastest, and update the cells on the boundary.

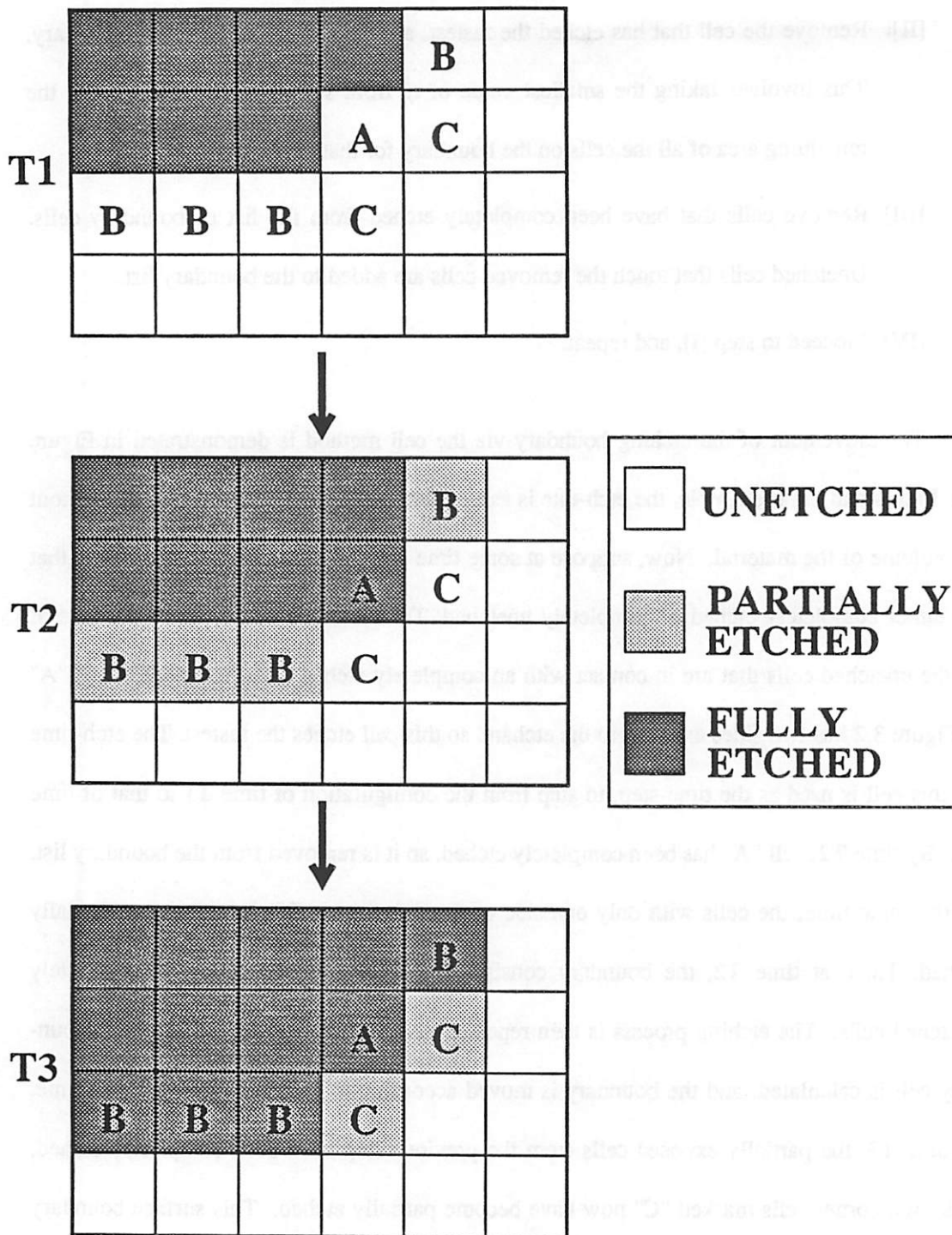
This involves taking the smallest value of  $t_r$  from step [I], and recalculating the remaining area of all the cells on the boundary for that particular value of  $t_r$ .

[III] Remove cells that have been completely etched from the list of boundary cells.

Unetched cells that touch the removed cells are added to the boundary list.

[IV] Proceed to step [I], and repeat.

The movement of the etching boundary via the cell method is demonstrated in Figure 3.2. In this particular example, the etch-rate is assumed to be constant and uniform throughout the volume of the material. Now, suppose at some time  $T1$ , the material consists of cells that are either completely etched or completely unetched. The etching boundary then consists of all the unetched cells that are in contact with an completely etched cell. At time  $T1$ , cell "A" in Figure 3.2 has two sides exposed to the etchant, so this cell etches the fastest. The etch-time for this cell is used as the time-step, to step from the configuration of time  $T1$  to that of time  $T2$ . By time  $T2$ , cell "A" has been completely etched, so it is removed from the boundary list. At the same time, the cells with only one side exposed (marked "B") have become partially etched. Thus, at time  $T2$ , the boundary consists of partially etched cells and completely unetched cells. The etching process is then repeated. Again, the time to etch for each boundary cell is calculated, and the boundary is moved according to the minimum cell etch-time. At time  $T3$ , the partially exposed cells from the previous step have been completely etched, while the corner cells marked "C" now have become partially etched. This surface boundary movement proceeds until the total etch-time equals or exceeds some targeted etch-time.



**Figure 3.2 :** The Cell Algorithm. The material is divided into cells, each fully etched, partially etched, or unetched. For each time step, the fastest etching cell is removed, and the area of the slower etching cells is updated.

### 3.3. IMPLEMENTATION OF THE CELL-REMOVAL ALGORITHM

The beauty of the cell approach is that it is computationally simpler than directly following the moving boundary of the material being etched. The cell-removal algorithm can be implemented simply by using an array of cells, each with flags denoting its etched or unetched condition. The cells on the boundary are placed in a list, together with data on the unetched (or etched) area of each cell in the list. Because of the rectangular structure of the cells, updating the boundary is simple; every time a cell is removed, its four neighboring cells are checked, and added to the boundary list if necessary.

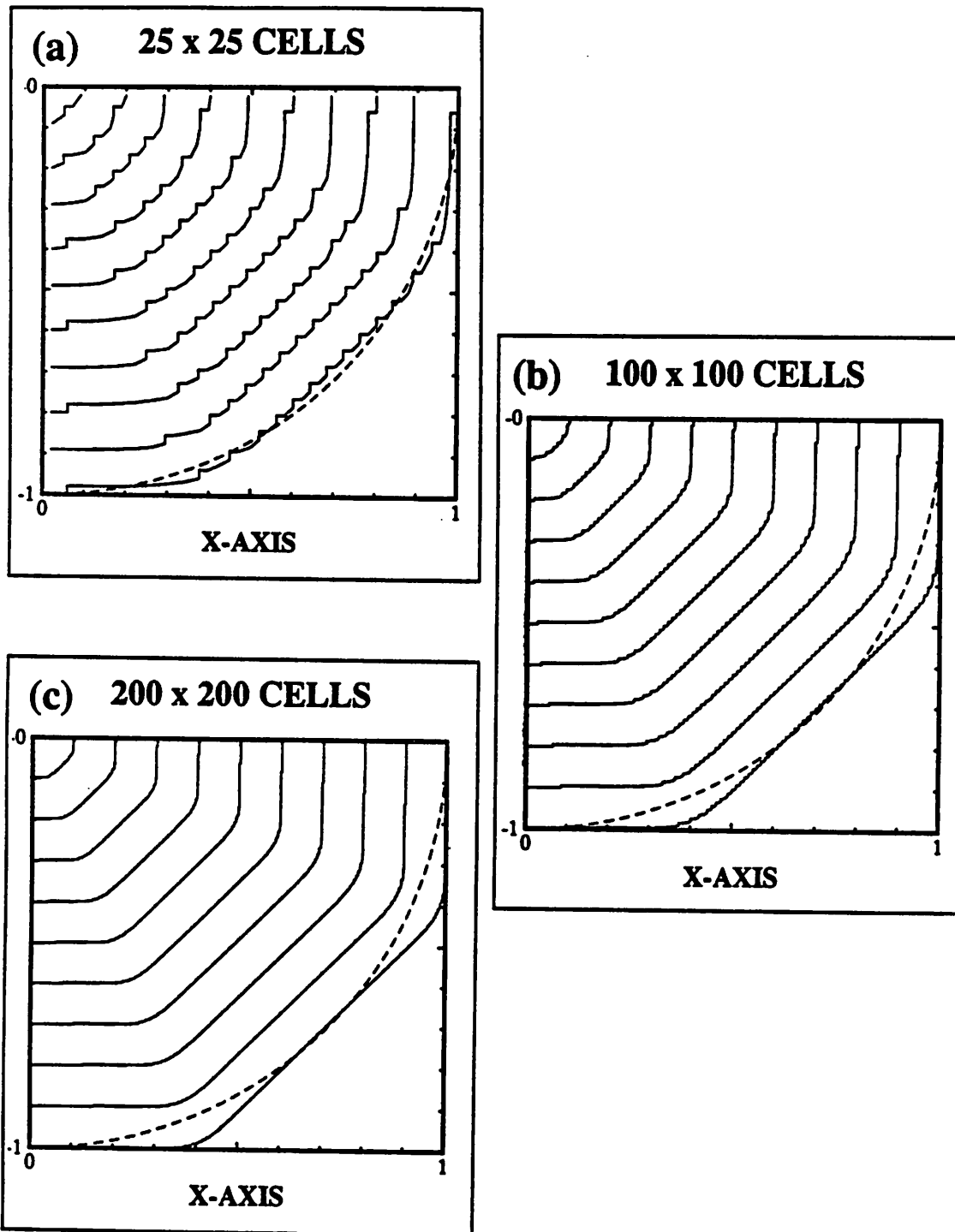
### 3.4. ACCURACY AND THE CELL-REMOVAL ALGORITHM

#### 3.4.1. Faceting during Uniform Circular Etching

Unfortunately, the cell model as described above does not produce accurate results; starting with a single etched cell, a two-dimensional uniform etch produces an octagon instead of a circle. Figure 3.3 shows uniform etch profiles for 3 different cell densities. When only  $25 \times 25$  cells are used in the  $1 \mu\text{m} \times 1 \mu\text{m}$  simulation, the profiles do look somewhat circular. The raggedness of the profiles are due to the relatively large cell width. † If the number of cells is increased to  $100 \times 100$ , the curves become smoother because of the smaller discretization. However, the etched profiles now show distinct facets in the horizontal, vertical and diagonal directions! Furthermore, the etch-profile becomes more octagon-like if the number

---

† The cell-removal algorithm keeps track of the area but not the shape of the cell that has been etched. This leads to uncertainty in the shape of the unetched material within the cell itself. For example, a cell with 50% etched material could have any of the shaded shapes shown in Figure 3.1. As a result, the accuracy of the cell algorithm depends on the width of each cell. In Figure 3.3, the profiles are drawn from the cells on the boundary list; contouring routines could be used to smooth out these curves.



**Figure 3.3:** 2D uniform etching beginning from a seed point at coordinates (0,0). The solid lines represent the cell-etched profiles at etching times of 0.1 - 1.0 seconds. The expected result, a semi-circle, is plotted in a dashed line. The simulations were run with (a) 25 x 25 cells, (b) 100 by 100 cells, and (c) 200 by 200 cells. The etch rate is 1  $\mu\text{m}/\text{sec}$ .



of cells is increased to  $200 \times 200$ .

This octagonal behavior is most likely due to the fact that each cell has only four neighbors. In fact, etching with the cell algorithm can be likened to etching a crystalline material with a cubic atomic lattice. It is well known that crystalline materials such as silicon will etch faster along certain crystal planes. (See, for example, Foote,<sup>7</sup> Bean.<sup>8</sup>) In a cubic atomic lattice, preferential etching will occur along the horizontal, vertical and diagonal planes, thus producing octagonal facets similar to those shown in Figure 3.3.

It is not easy to eliminate or decrease the faceting problem in cell-etching. Increasing the number of cells in the material does not help. As mentioned previously, the facets become sharper as the number of cells are increased. This is probably because as the number of cells are increased, the material becomes more similar to a cubic atomic lattice, and crystal plane etching dominates. One could try to increase the number of neighbors of a cell, perhaps by using a hexagonal cell structure. In this case, however, there will still be preferred directions of etching perpendicular to the hexagon surfaces. Again, facets will be formed. Another method that might work is to subdivide each cell, and to keep track of the area and shape of the material that has been etched. This method, however, greatly increases the difficulty in implementing the cell algorithm. Another interesting "fix" uses a spill-over technique, in which the etch is "spilled over" to neighboring cells. This method, used in the *SOLID<sup>5</sup>* simulation program, will be described in a later section.

### 3.4.2. Photoresist Etching

An example of etching photoresist using the cell-removal method (without correction for faceting) is shown in Figure 3.4. Also plotted are the results from *SAMPLE*, in which the

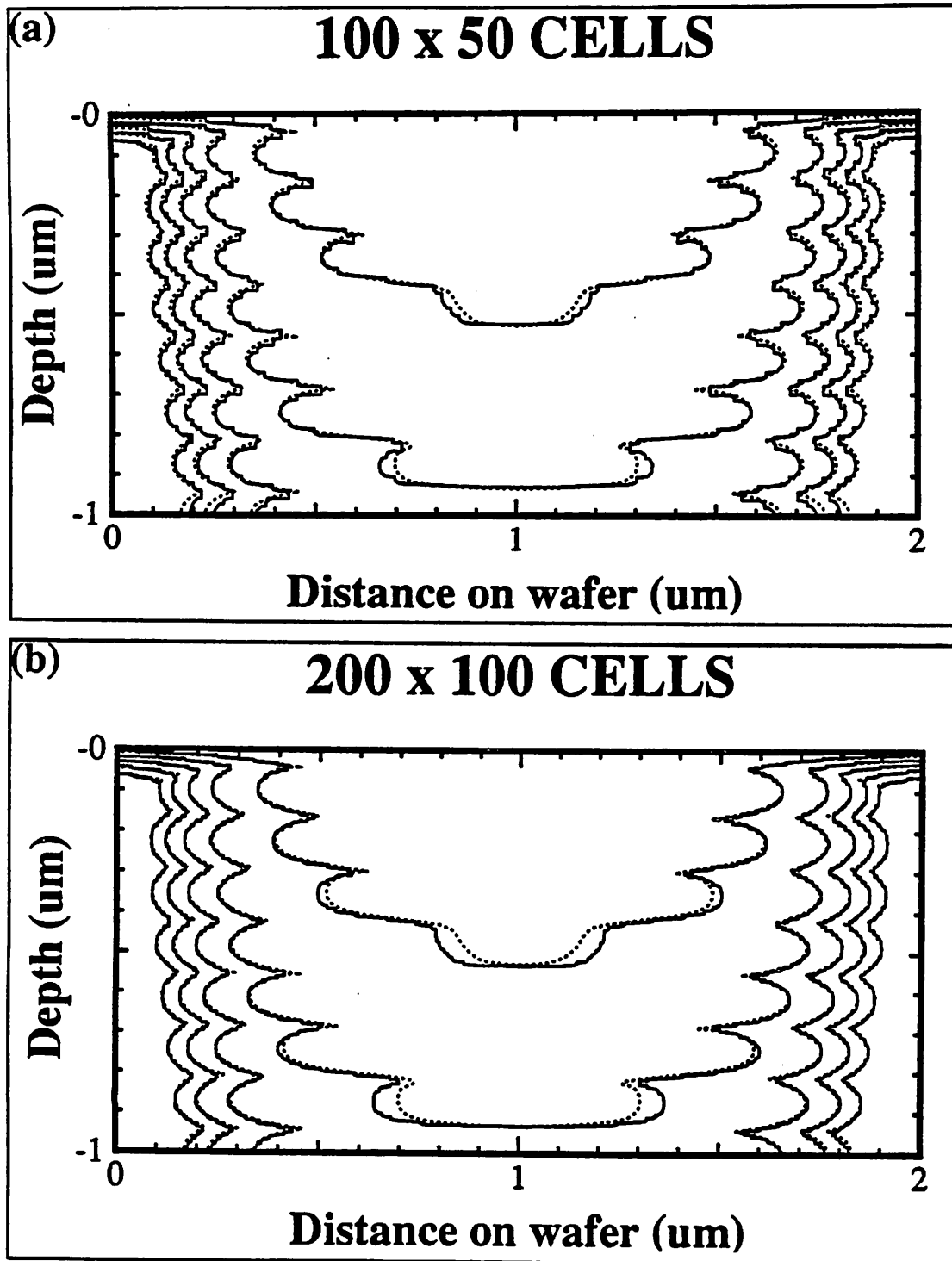


Figure 3.4 : Resist profiles simulated using the cell-removal algorithm, at development times of 15, 30, 45, 60 and 75 seconds. The simulations use the etch-rate distribution plotted in Figure 2.3, with (a) 100 by 50 cells, and (b) 200 by 100 cells. The dotted lines are the profiles from the *SAMPLE* string simulation (Figure 2.4).

string method is used to advance the etch surface.† It is quite clear that the agreement between the cell and string methods increases as the number of cells is increased. At development times of 45, 60, and 75 seconds, the 20,000-cell and string profiles agree quite well; when only 5,000 cells are used, the agreement is not as good. This is perhaps to be expected. A smaller cell size leads to smaller discretization error, and thus to greater accuracy. In the  $100 \times 50$  cell simulation, each cell is  $0.02 \mu\text{m}$  wide; thus the accuracy of the simulation is within  $0.02 \mu\text{m}$ . When the cell density is increased to 100 cells per micron, the accuracy increases too, to within  $0.01 \mu\text{m}$ .

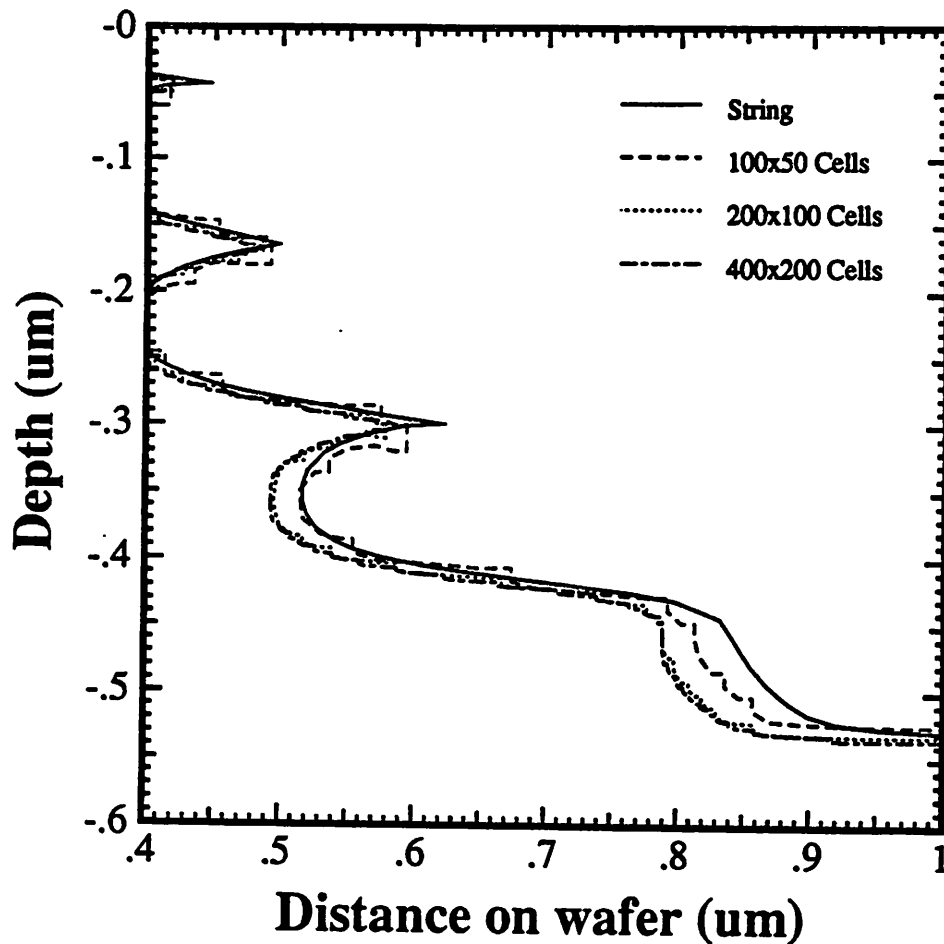
At the earlier development times of 15 and 30 seconds, however, the cell and string profiles disagree, especially at the bottom-most standing wave. It is somewhat discomfoting to find that the disagreement gets worse when the number of cells is increased from 5,000 to 20,000. However, the etch profiles do converge as more cells are used in the simulation. Figure 3.5 shows a blow-up of the resist profile at a development time of 15 seconds. The four curves show simulated profiles using the string method and the cell method, with  $100 \times 50$  cells,  $200 \times 100$  cells, and  $400 \times 200$  cells respectively. The 20,000-cell and 80,000-cell profiles agree with each other, indicating that the simulation has converged. However, this convergent cell-profile is definitely not the same as the string profile; at the bottom-most standing wave, the cell-etched profile is approximately  $0.05 \mu\text{m}$  wider than the string-etched profile.

Unfortunately, it is not really possible to point out which simulation profile is "correct", as the actual resist profile is not known. However, it is possible to draw conclusions on accuracy based on convergence. The string algorithm converges to the very same solid-line profile

---

† The string etching algorithm is described in Chapter 6.

## 15 sec Development Time



**Figure 3.5 :** Simulated resist profiles at a development time of 15 seconds. The simulations use the etch-rate distribution plotted in Figure 2.3. The different curves represent simulations using different etching algorithms.

shown in Figure 3.5 as the string segment length is decreased. Further, as will be shown in Chapter 7, the ray algorithm, in which the etch differential equation is actually solved (albeit discretely), also produces the same profile. Thus, there is greater confidence in the string-etched profile than in the cell-etched resist profile.

The effects of faceting also have to be considered. In the uniform etch simulations shown in Figure 3.3, the cellular structure causes the etch-profile to overshoot the theoretical circular profile, with the maximum overshoot occurring  $22.5^\circ$  from the horizontal and vertical axes. This increase in the effective etch-rate in certain directions could also be a factor affecting the final etch profile. The strong octagonal facets displayed in uniform etching do not show up in the resist simulations of Figure 3.4 or 3.5, probably because the faceting has been averaged out by the non-uniform etch-rate throughout the volume of the resist. However, it is quite possible that the effective etch-rate has been increased in certain places, due to the cellular faceting problem. Also, recall that in the uniform circular etch, the facets on the profiles became sharper as more cells were used. This effective increase in etch-rate with increasing cell numbers could account for the fact that in the resist simulations, the 5,000-cell profile is closer to the string profile than the 20,000-cell profile is. Thus, faceting could be causing faster-etched profiles at the 15 and 30 second development times.

### 3.5. COMPUTATION TIME

Another drawback of the cell-removal algorithm is that the cell method is quite slow. To advance the boundary layer, it is necessary to compare the etch-time of *every* cell in the boundary list. The time-step for moving the boundary is controlled by the minimum etch-time of the fastest etching cell in the boundary list, so it could take a long long time to reach a desired etch-time if there were a lot of cells in the material. In two dimensions, the

computation time quadruples as the number of cells per dimension is doubled. The number of cells in the boundary list also increases at the same time, adding another multiplicative factor of two to the computation time. So, in a 2D cell approach, the computation time goes roughly as  $O(N^3)$ , where  $N$  is the number of cells in one dimension. The computation times for the 15-second development simulation in Figure 3.5 are shown in Table 3.1. These computation times were measured on a SUN 4/280 (0.8 MFLOPS, 10 MIPS). †

---

Number of Cells	Computation Time	CPU Increase
$100 \times 50 = 5,000$	22 sec.	1.0
$200 \times 100 = 20,000$	141 sec.	6.4
$400 \times 200 = 80,000$	1058 sec.	$6.4 \times 7.5$

Table 3.1 : Cell-Removal Algorithm : Computation time on a SUN 4/280

---

The computation time increases by a factor of 48 when the number of cells is increased from 50 per micron to 200 per micron. The  $O(N^3)$  formula predicts an increase of  $4^3 = 64$ , which is not too far off. Nevertheless, an 80,000-cell simulation takes up 1058 seconds, or almost 18 minutes of CPU time. By comparison, the *SAMPLE* string simulation uses only some 13 seconds on the same machine!

The general thrust of research in the cell algorithm has been to increase the speed of the cell method without spending an excessive amount of computer memory. One approach for speeding up the computation is to track the area of the cells with integers instead of floating-point variables. Computers can typically handle a higher rate of integer instructions compared to floating-point operations, so the use of integer mathematics confers a considerable speed advantage (approximately 3-10 times faster). Another method for increasing the computation

---

† MFLOPS = Millions of Floating Point Operations per Second, MIPS = Millions of (integer) Instructions per Second

speed is to modify the method of advancing the surface. Recall that in the original cell algorithm, the time-step for moving the boundary is controlled by the minimum etch-time for all the cells currently on the boundary. So, at every advancement, the etch-time of each cell must be compared. If, however, the surface could be moved with a constant time-step, it will no longer be necessary to compare the etch-speed of each cell in the boundary, and the computation can proceed at a faster rate. The constant time-step scheme, which has been used by Toh (Chapter 4) and Pelka,<sup>5</sup> decreases the 2D computation time from roughly  $O(N^3)$  to  $O(N^2)$ ,  $N$  being the number of cells per dimension.

Pelka's<sup>5</sup> cell-removal scheme, implemented in 3D in the *SOLID* simulator, is a particularly interesting approach that combines both the constant time-step with integer cell calculations. In this "spill-over" method, each cell is made up of an integer number of microcells. The surface is advanced by subtracting a certain number of microcells from each cell on the surface; the actual number of microcells removed is proportional to the local etch-rate multiplied by the constant time-step. If the etch uses up more microcells than are currently available in the active cell, the etch is allowed to spill-over to neighboring cells, and spill-over microcells are subtracted from the neighboring cells.

The spill-over cell method is impressively fast especially for a 3D cell-based simulation. The program executes with CPU time between 2 and 20 minutes on a VAXstation 3540, which seems to be a parallel multiprocessor machine.<sup>9</sup> It is very likely that the computation time required for the 3D etch simulation would be much greater than 20 minutes on a typical serial single-processor workstation. At the same time, the spill-over technique is also expensive in its needs for memory. The *SOLID* brochure states that fairly large memory (24-32MB) is required for optimum 3D performance. The spill-over technique is supposedly also more accurate than Dill's cell algorithm. Pelka claims that facets are not encountered in

circular or spherical etching. Nevertheless, the accuracy of the spill-over technique has not yet been established conclusively. There is still some doubt over the validity of the spill-over method for handling over-etched cells. Another area of concern is with etching situations where the etch-rate has a high aspect ratio, that is, the etch-rate varies from very large values to very small values throughout the volume of the material. In areas of small etch-rate, the surface will not move if the etch-rate is smaller than 1 microcell per time-step. This could lead to discretization errors that could in turn affect the accuracy of the simulation.

### 3.6. THE CELL-REMOVAL ALGORITHM IN 3D

The cell algorithm is quite easy to implement. However, its primary drawback is the lack of accuracy in the simulation results. The cell algorithm becomes more accurate as the number of cells in the simulation volume are increased. This is because the discretization error decreases with cell width. However, at the same time, when the number of cells is increased, faceting becomes more of a problem, producing incorrect profiles in certain locations. It does not help that it is difficult, if not impossible, to predict which parts of the profile are affected by faceting. Thus, when the cell density is increased, the discretization error decreases and accuracy increases, but faceting increases, and the accuracy decreases. The overall result is a lack of confidence in the accuracy of the simulations.

The advantages and disadvantages of the cell algorithm in 2D apply also in 3D. The addition of an additional dimension pushes the computation time up further to roughly  $O(N^4)$  or even  $O(N^5)$ , where as before,  $N$  is the number of cells in one dimension. Using both constant time-steps and integer calculations, the computation time can be speeded up to at best  $O(N^3)$ . Implementing the cell algorithm in 3D is also more difficult, because of the need to keep track of a cell's 6 neighbors, as compared to only 4 neighbors in 2D. Furthermore,



memory becomes a limiting factor.  $200 \times 200 \times 200$  cells will consume some 16 Mbytes of memory, assuming optimistically that each cell only uses 2 bytes of memory. The large amount of memory and computation time needed for the 3D cell method makes it quite unsuitable for non-mainframe applications. And of course, the accuracy, or lack of it, also counts against the cell method.

## REFERENCES

1. F.H. Dill, A.R. Neureuther, J.A. Tuttle, E.J. Walker, "Modeling Projection Printing of Positive Photoresists," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 456-464, July 1975.
2. F. Jones, J. Paraszczak, "RD3D (Computer Simulation of Resist Development in Three Dimensions)," *IEEE Transactions on Electron Devices*, vol. ED-28, no. 12, pp. 1544-1552, December 1981. *RD3D* : 3D Cell method.
3. Y. Hirai, M. Sasugo, M. Endo, K. Ikeda, S. Tomida, S. Hayama, "Three Dimensional Process Simulation for Photo and Electron Beam Lithography and Estimations of Proximity Effects," *Symposium on VLSI Technology*, p. 15, 1987. 3D Cell method.
4. J. Bauer, W. Mehr, U. Glaubitz, "Simulation and Experimental Results in 0.6  $\mu\text{m}$  Lithography using an I-Line Stepper," *Proceedings of SPIE : Optical/Laser Microlithography III*, vol. 1264, pp. 431-445, March 1990. *LITHSIM* : 3D Cell method.
5. J. Pelka, "SOLID : Comprehensive Three Dimensional Simulation Program for Optical Microlithography," *Information Brochure, Fraunhofer-Institut fur Mikrostrukturtechnik*, May 1990. *SOLID* : 3D Cell method.
6. M. Fujinaga, N. Kotani, T. Kunikiyo, H. Oda, M. Shirahata, Y. Akasaka, "Three-Dimensional Topography Simulation Model : Etching and Lithography," *IEEE Transactions on Electron Devices*, vol. ED-37, no. 10, pp. 2183-2192, October 1990. *3D-MULSS* : 3D Cell method.
7. W.F. Foote, "Two-Dimensional Silicon Etching ," *M.S. Thesis*, University of California, Berkeley, 1989.
8. K.E. Bean, "Anisotropic Etching of Si," *IEEE Transactions on Electron Devices*, vol.

ED-25, p. 1185 , 1978.

9. J. Pelka, *Simulation of Ion-Enhanced Dry-Etch Processes*. Presented at the SPIE Fall Meeting 1990, Santa Clara, CA.

## CHAPTER 4

### THE MODIFIED CELL METHOD

#### 4.1. INTRODUCTION

As discussed in Chapter 3, the cell-removal algorithm is an atomistic approach, where each cell/atom is removed as it comes into contact with an etchant. However, this atomistic model also produces facets, which should not be present in an etch of a uniform homogeneous material. In addition, the cell-removal algorithm is slow and requires a considerable amount of computer memory. But the cell-removal algorithm is easy to implement and is very robust as well. It can easily handle simulation boundaries and underlying topography, and does not have the looping problems encountered in surface-advancement algorithms. For these reasons, the cell algorithm remains an active area of research.

One area in which the cell-removal algorithm could be improved is the method of the etch-front advancement. This chapter describes a constant time-step method in which the Huygens principle is used to advance the etch-front. The improvement in accuracy and the role of etch discretization on the accuracy of the cell-based simulation are considered.

#### 4.2. THE HUYGENS PRINCIPLE APPLIED TO CELL-ETCHING

The faceting problem encountered in Dill's cell algorithm seems to be directly related to the treatment of the cells as atoms arranged in cubic lattices. Fortunately, a better etching model can be derived from the Huygens principle. † This theorem asserts that each element on a propagating wavefront may be regarded as the center of a secondary disturbance which gives

---

† Born & Wolf,<sup>1</sup> Chapter 3, p.132.

rise to spherical wavelets. Moreover, the position of the wavefront at any later time is the envelope of all such wavelets. The Huygens principle is actually used to describe the diffraction of light, but it can be applied quite nicely to isotropic etching too. Here, the etching proceeds along an etch front, and each point on the etch front is an etching center, "radiating" etch wavelets. The etch front at some later time is the envelope of these etch wavelets.

The modified cell algorithm presented in this chapter is a new and novel etching simulation method based on the Huygens principle. The volume of the material is divided into tiny cells, each either etched or unetched. Just like the cell-removal method, etching proceeds along a surface in contact with the etchant. But unlike the usual cell-removal method where only the nearest neighbors are removed, in the modified version, the cells are removed using the Huygens principle applied to each cell on the etch boundary. All the cells within the radius of a Huygens etch front centered at a cell on the boundary are removed. After each time-step, the boundary is updated; the boundary cells are those etched cells that are in contact with unetched cells.

### **4.3. THE ALGORITHM FOR CELL-REMOVAL**

The modified cell simulation begins with the selection of a constant time-step based on the etch-rate configuration. During every time-step, boundary cells are swept, and new boundary cells are determined. The algorithm can be stated as follows :

- [I] For each boundary cell :
  - (a) Determine the local etch-rate at the center of the boundary cell. The radius of influence of the boundary cell is the local etch-rate multiplied by the (constant) time-step.

(b) Modify the etch flags of all the cells within the radius of influence. Unetched cells inside this radius become etched.

[II] Find the new boundary cells; these are etched cells next to unetched cells.

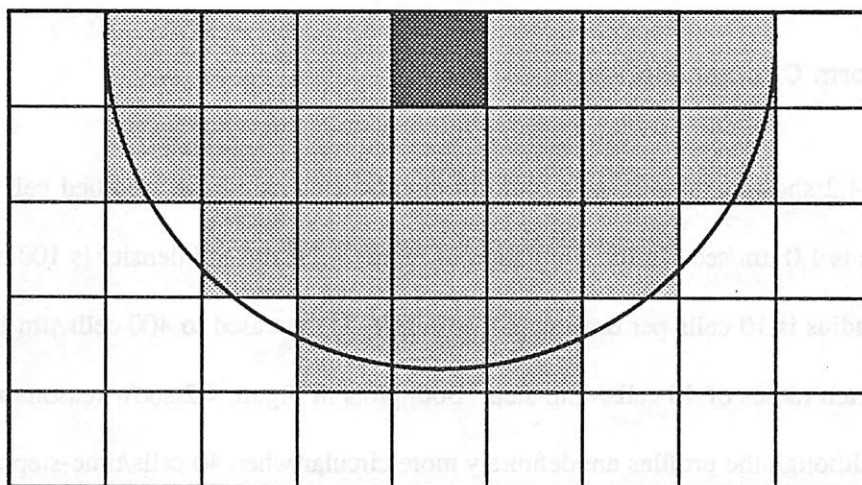
[III] Proceed to [I] and continue until the total etch-time has been reached.

Figure 4.1 shows examples of etching with the modified cell method. In Figure 4.1a, there is only a single etched cell, which also is a boundary cell. Assuming that the local etch-rate is 3.5 cells per time-step, all the cells within the solid circle shown in Figure 4.1a will become etched during the next time-step. The boundary cell list now contains all the cells on the outer perimeter of the circle. Figure 4.1b shows another example, this time with multiple boundary cells. Again, the etch-rate is 3.5 cells per time-step. During the time-step, all the cells within the radius of influence of each boundary cell become etched. The etch front is then the envelope of all the overlapping circles.

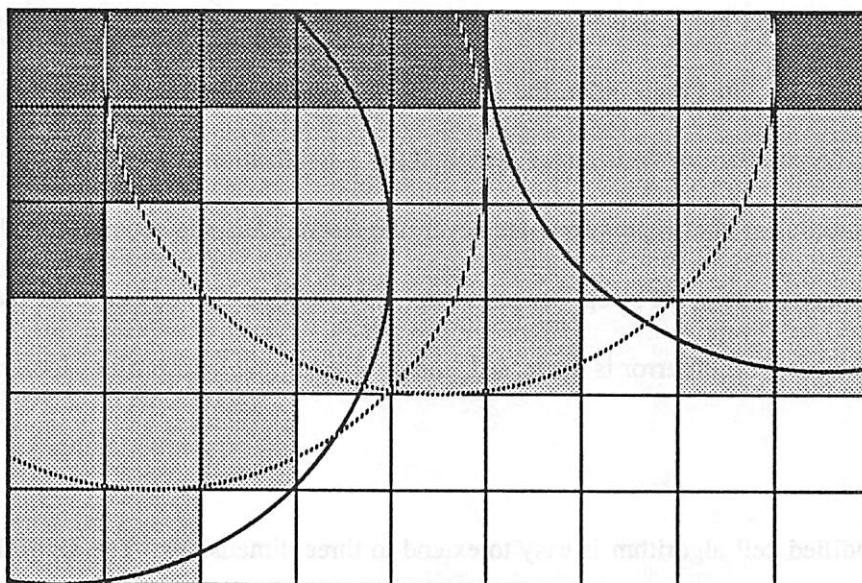
#### 4.4. IMPLEMENTATION OF THE MODIFIED CELL-REMOVAL ALGORITHM

The modified cell algorithm is very easy to implement in either 2D or 3D. This algorithm merely requires an array of integer flags to keep track of the etched (or unetched) state of each cell in the volume of the material. At the same time, a list of all the boundary cells has to be maintained. This can best be done with a linked list. During each time-step, the local etch-rate at a boundary cell is multiplied by the time-step to find the cell's radius of influence. All the cells with centers within this radius become etched. After each time-step, all the cells in the material are scanned; etched cells that have neighboring unetched cells are added to the boundary list.

(a) *SINGLE BOUNDARY CELL*



(b) *MULTIPLE BOUNDARY CELLS*



**Figure 4.1 :** The modified cell method. All the cells within the radius of influence of the boundary cells are etched.

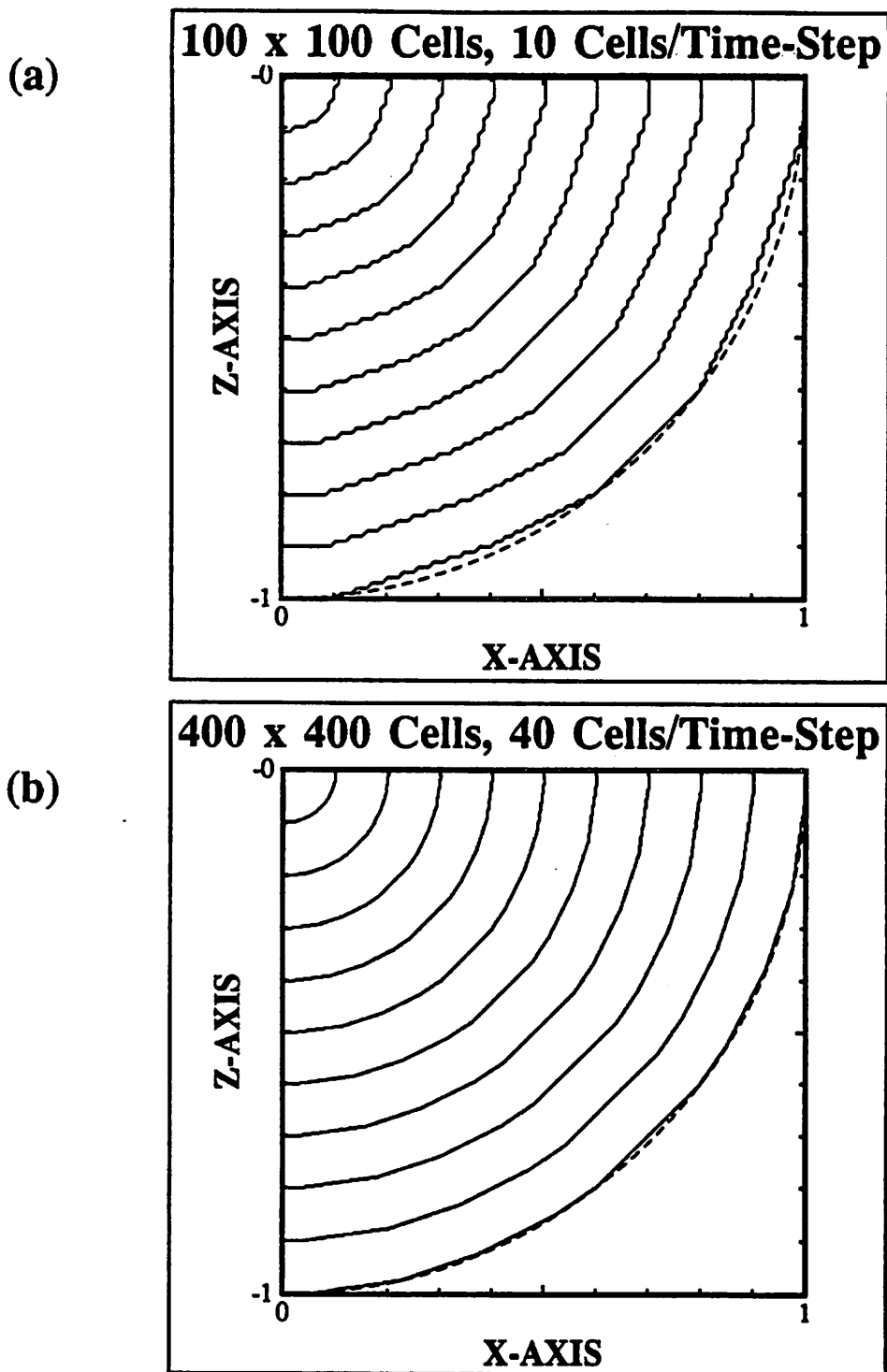
## 4.5. TESTING THE MODIFIED CELL-REMOVAL ALGORITHM

### 4.5.1. Uniform Circular Etching

Figure 4.2 shows examples of a uniform circular etch using the modified cell method. The etch-rate is  $1.0 \mu\text{m}/\text{sec}$ . In the simulation of Figure 4.2a, the cell density is  $100 \text{ cells}/\mu\text{m}$ , so the etch-radius is 10 cells per time-step. The density is increased to  $400 \text{ cells}/\mu\text{m}$  in Figure 4.2b, for an etch radius of 40 cells/time-step. Both plots in Figure 4.2 show reasonably circular profiles, although the profiles are definitely more circular when 40 cells/time-step are used. Nevertheless, facets are formed on the circular etch profiles. This is actually quite understandable given the nature of the etch algorithm. The problem actually can be stated as follows : how many squares does one have to use to fill a circle? As Figure 4.1a shows, a circle with a radius of 3 cell-widths is not very circular at all; in fact, the cellular circle of Figure 4.1a looks more like an octagon! But as more and more cells are packed into the circle, the cellular shape looks more and more like a circle. The same reasoning applies for the simulations shown in Figure 4.2. In Figure 4.2a, the first time-step creates a circle with a radius of 10 cell-widths. The circular profile is discretized, and the discretization error propagates outward to affect the profiles at later time-steps. However, the discretization is decreased with greater cell density. When 40 cells are swept per time-step, the first circle etched out looks reasonably circular. The discretization error is decreased, and as a result, the later profiles look more circular.

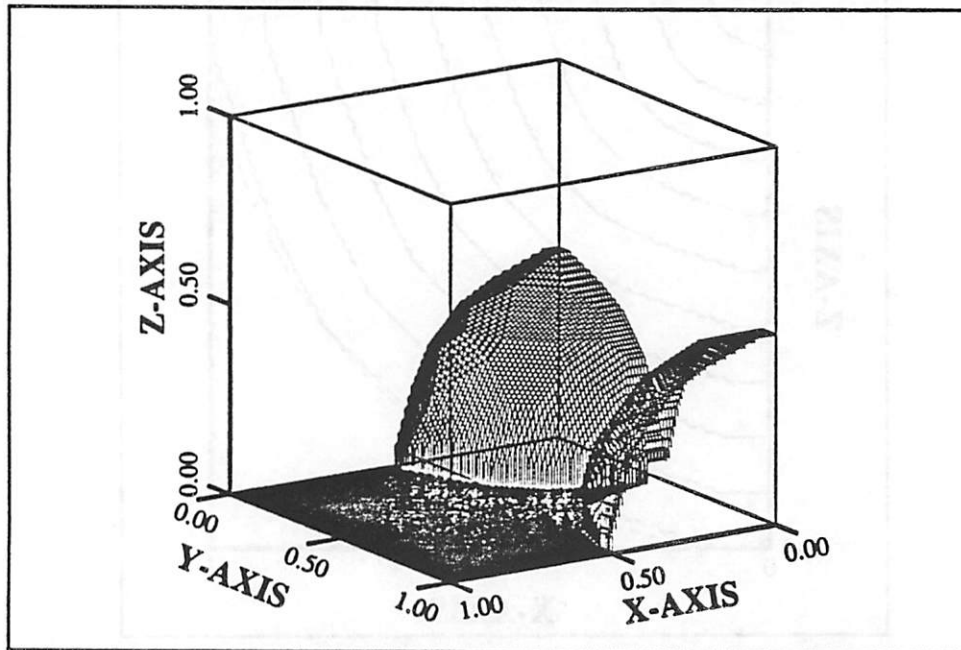
The modified cell algorithm is easy to extend to three dimensions. The algorithm basically remains unchanged, except that to find all the cells within the radius of influence of the boundary cells, a Huygens sphere is used instead of a circle. Figure 4.3 shows examples of the uniform circular etch in 3D. The etch begins from two seed points at coordinates (0,0,0)





**Figure 4.2 :** 2D uniform etching beginning from a seed point at coordinates (0,0). The solid lines represent the cell-etched profiles at etching times of 0.1 - 1.0 seconds. The expected result, a semi-circle, is plotted in a dashed line. The simulations were run with (a) 10 cells per time step, and (b) 40 cells per time-step. The etch rate is 1  $\mu\text{m}/\text{sec}$ .

(a) 3D UNIFORM ETCHING : 5 TIME STEPS



(b) 3D UNIFORM ETCHING : 10 TIME STEPS

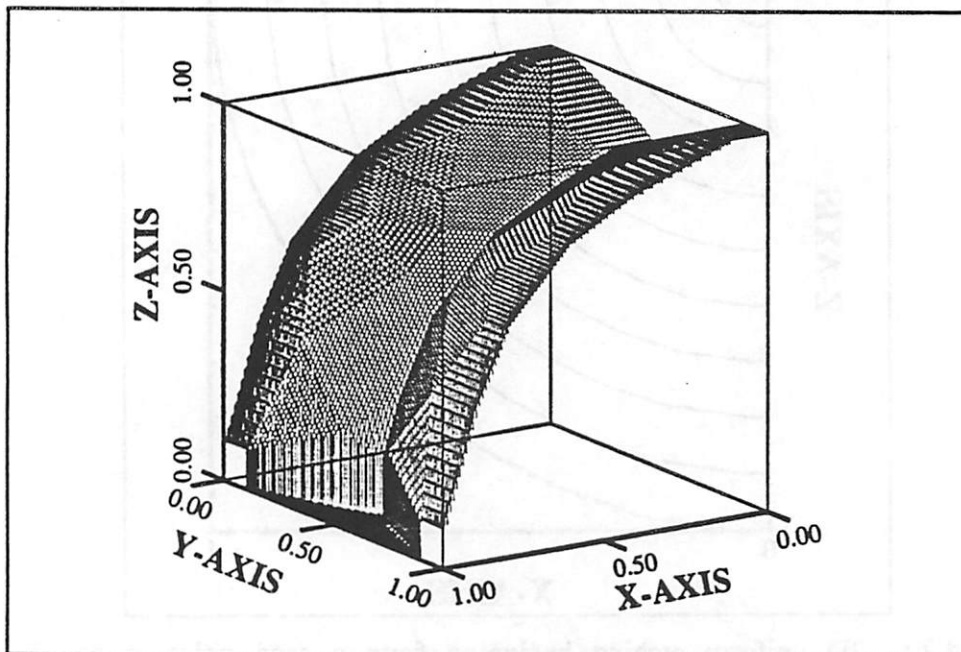


Figure 4.3 : 3D uniform etching beginning from two seed points at coordinates  $(0,0,0)$  and  $(0,1,0)$ . The cell-etched profiles are shown at etch times of (a) 0.5 sec, and (b) 1.0 sec. The etch rate is 1  $\mu\text{m}/\text{sec}$ , and the time-step is 0.1 sec. The etch radius is 10 cells per time-step.

and (0,1,0), and proceeds outwards in time-steps of 0.1 seconds. The etch-rate is uniform, and as in the previous example, has a value of 1  $\mu\text{m}/\text{sec}$ . The simulation cube consists of  $100 \times 100 \times 100 = 10^6$  cells, and the etch radius is 10 cells per time-step. The etched profile after 0.5 seconds (5 time-steps) is shown in Figure 4.3a. As can be seen, two spheres are formed, both with 0.5  $\mu\text{m}$  radius. Both of these spheres do have multiple facets, but as in the 2D case, these facets are formed due to the errors from the cellular discretization of the spheres.

Figure 4.3b shows the profile of the etch front after 10 time-steps, or 1 second of etching. The two spheres have continued to etch outward at a uniform pace. Both spheres are still faceted. In addition, the two spherical etch fronts have intersected and merged. It is important to note that no loops have formed at the intersection. The intersection of the two spheres is very clean and tidy, as would be expected of a volume etching algorithm. As shall be seen in the following chapters on surface-advancement algorithms, this same etching case results in a looped intersecting mesh when either the ray or string algorithms are used to simulate the etching.

It is possible to pack more cells into the simulation volume in order to reduce faceting. But computers do have an upper limit on memory. The  $10^6$  cells used in this particular simulation use about 2 bytes each; if  $400 \times 400 \times 400$  cells were used for an etch radius of 40 cells per time-step, some  $2 \times 400^3$  or 128 MBytes of memory would be required.† This is beyond the capabilities of most engineering workstations. But even the 1,000,000 cell simulation consumes some 2 MBytes of memory and a few hours of computation time. The 5-time-step simulation of Figure 4.3a took approximately 20 minutes on a SUN4/280, while the 10-time-

---

† Note though that these numbers are based on 2 bytes for each cell in the 3D array. Actually, only 2 bits need to be used for the two integer flags at each cell. Thus, the memory can be reduced by a factor of 8. But in the C programming language, it is not easy to manipulate bits. To save coding time, the code was implemented using integer flags instead of bit flags.

step simulation lasted approximately 4 hours.

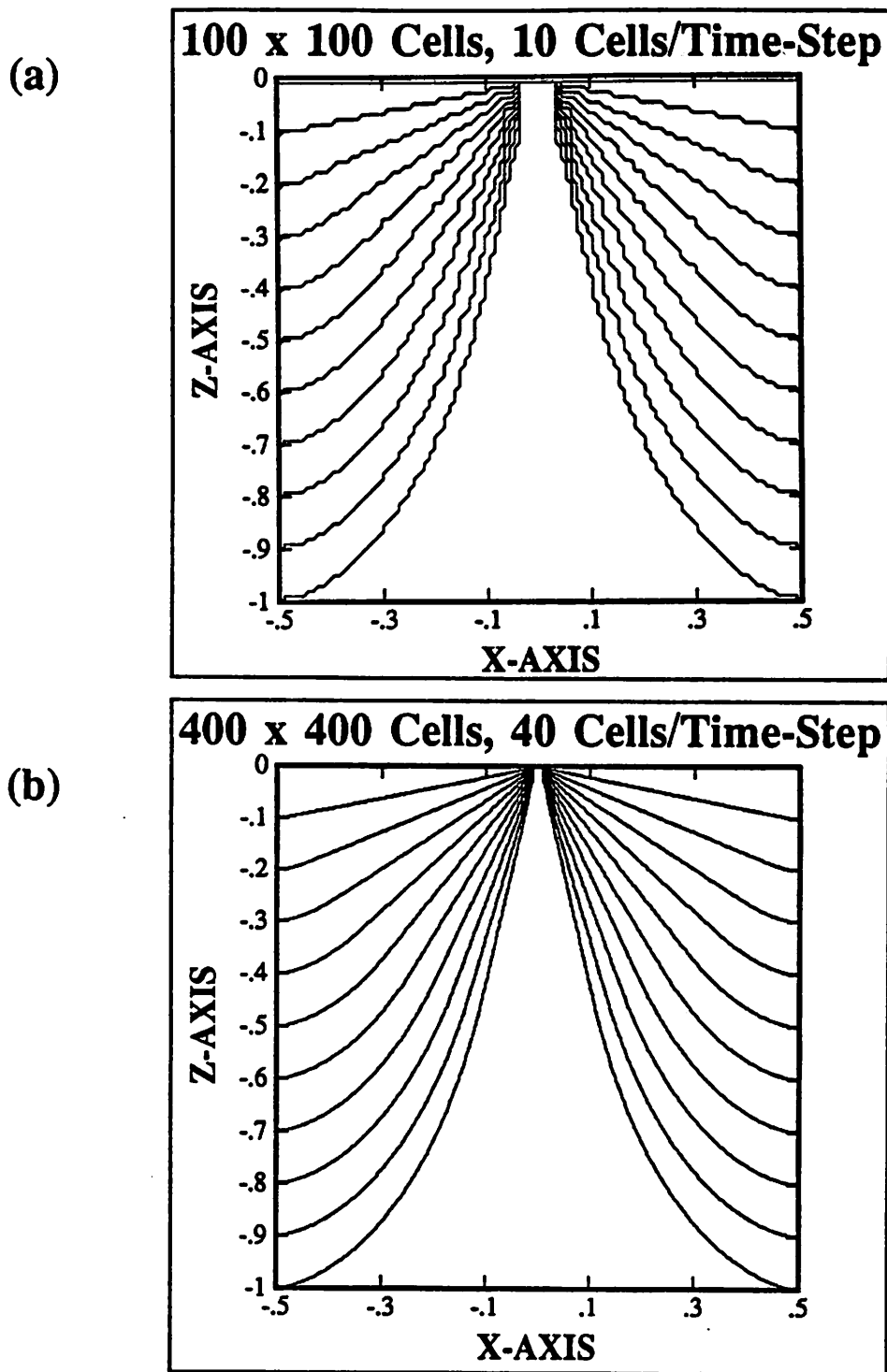
#### 4.5.2. Etching with Triangular Analytic Functions

So far, the modified cell algorithm looks quite promising. The behavior of the modified cell algorithm may be analyzed further using a variety of analytical etch-rate functions. One such useful function is a triangular etch-rate with the equation

$$R(x,y,z) = 2|x| \mu\text{m/sec}, \quad |x| < 0.5 \quad [4.1]$$

Figure 4.4 shows the profiles produced at 0.1 second intervals from the above triangular etch-rate function. In Figure 4.4a, the etch is fastest at the edges corresponding to  $x = -0.5$  and  $x = 0.5$ . The etch radius is 10 cells per time-step at these edges, and decreases linearly with decreasing  $|x|$ . In Figure 4.4b, the maximum etch-radius is 40 cells per time-step. A comparison of the two plots in Figure 4.4 shows that the etched profiles are quite similar for larger values of  $|x|$ . The profiles of Figure 4.4b are smoother, but this can be attributed to the increased cell density used in the simulation.

The behavior of the profiles for small values of  $|x|$  is particularly interesting. When 10 cells per time-step are used, the etch depth (vertical distance from the  $z = 0$  axis) is zero for  $x = 0$ , remains zero for small values of  $|x|$ , and then suddenly jumps to a non-zero value. The sudden step is more noticeable in the 10 cell/time-step simulation. This fascinating behavior is actually caused by discretization. If the etch-rate is small, and the etch radius is less than one cell-width, then no cells are etched regardless of the total etching time spent. To illustrate, suppose the etch-rate is 1 unit per time-step, and the cell is 1 unit wide, so that the etch radius is one cell-width. At every time-step, the etch circle emanating from the center of the boundary cell is large enough to touch the center of the boundary cell's neighbor. This



**Figure 4.4 :** 2D etching using the triangular etch-rate function of Equation [4.1]. The solid lines represent the cell-etched profiles at etching times of 0.1 - 1.0 seconds. The simulations were run with (a) 10 cells per time step, and (b) 40 cells per time-step. The maximum etch rate is 1  $\mu\text{m}/\text{sec}$ .

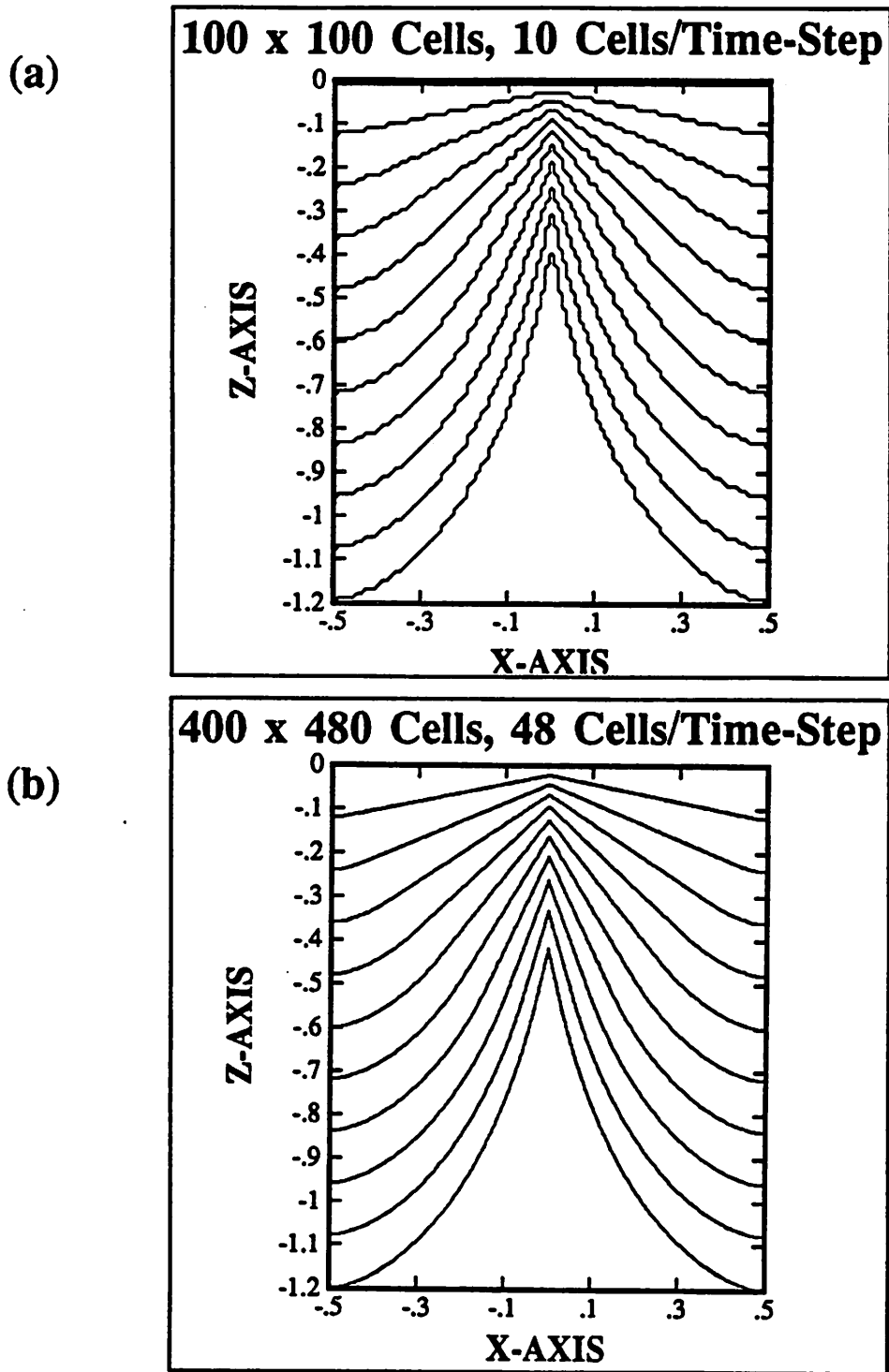
neighboring cell then becomes etched, and the etch proceeds at 1 cell per time-step. But now, suppose the etch-rate is a mere 0.9 units per time-step, so that the etch radius is only 0.9 cell-widths. Now, the etch radius does not hit the center of the boundary cell's neighbor, and as a result, the neighbor remains unetched. This continues for all subsequent time-steps, and the boundary remains unchanged.

Thus, one weakness of the modified cell algorithm is that it is sensitive to discretization. Because the algorithm does not keep track of the etched area of the cell, discretization errors occur when the etch radius is not an integer multiple of the cell width. The discretization error is of particular importance when the etch radius is comparable to the cell width. This typically happens when the etch-rate varies widely in range, as was the case in the triangular etch-rate function.

Another interesting test function is that of a triangular function with a constant offset.

$$R(x,y,z) = 2|x| + 0.2 \mu\text{m}/\text{sec}, \quad |x| < 0.5 \quad [4.2]$$

This function actually produces a loop when either the ray or string algorithm is used to simulate etching. Figure 4.5 shows the simulated profiles for 12 cells/time-step and 48 cells/time-step respectively. The two plots are similar, and as in the previous example, the curves become smoother when the etch-radius includes a larger number of cells. But notice that at the center, at  $x = 0$ , the etch has proceeded to a depth of  $-0.4 \mu\text{m}$  after 1 second of etching. However, according to Equation [4.2], the etch-rate is only  $0.2 \mu\text{m}/\text{sec}$  at the center. After 1 second of etching, the profile should be only at  $-0.2 \mu\text{m}$ . So, why has the simulated etch gone further than predicted by the center etch-rate? The answer lies in the two-dimensional nature of the etch. In the center, the etch is dominated by off-center elements. The etch-rate is larger away from the center, so the off-center points will etch away areas that have not yet been reached by the center etch elements. As a result, etching proceeds faster than would be



**Figure 4.5 :** 2D etching using the triangular etch-rate function of Equation [4.2]. The solid lines represent the cell-etched profiles at etching times of 0.1 - 1.0 seconds. The simulations were run with (a) 12 cells per time step, and (b) 48 cells per time-step. The maximum etch rate is 1.2  $\mu\text{m}/\text{sec}$ , and the minimum etch rate is 0.2  $\mu\text{m}/\text{sec}$ .

expected from a simple one-dimensional vertical etch.

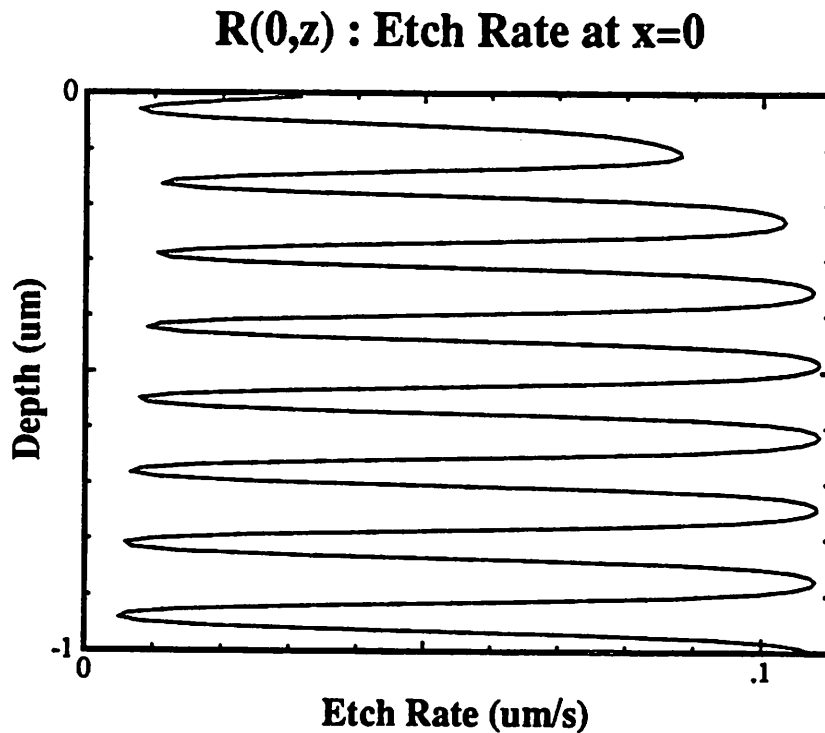
It is also noteworthy that the discretization error that showed up in the previous example does not seem to have occurred in this particular example. This is probably because the smallest etch radius in the  $100 \times 120$  cell simulation is relatively large compared to the cell width, covering 2 cells per time-step.

### 4.5.3. Photoresist Etching

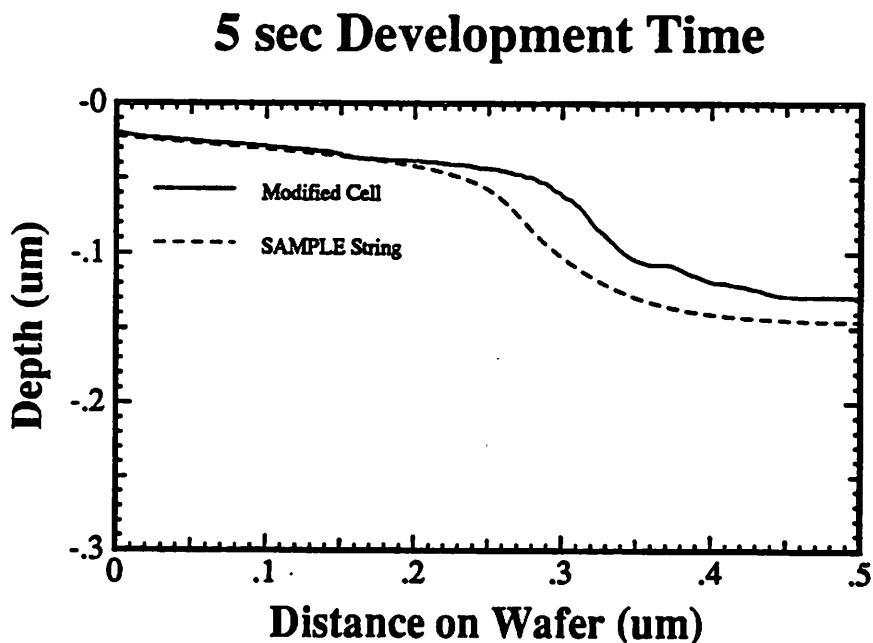
The discretization error in the modified cell algorithm makes photoresist etching very difficult. Typically, multiple reflections cause standing waves to form in the photoresist when the resist is exposed to light. As a result, the etch-rate fluctuates widely with depth into the resist. Figure 4.6 shows the etch-rate profile versus depth at the center of a large isolated space. This rate-vs-depth profile is actually taken from a vertical cut across the etch-rate contours shown in Figure 2.3. As shown in Figure 4.6, the etch-rate changes from approximately  $0.009 \mu\text{m}/\text{sec}$  to  $0.11 \mu\text{m}/\text{sec}$  in a sinusoid-like manner. Now, the etch radius in the modified cell algorithm depends on the incremental time-step chosen for the algorithm. With a time-step of 0.1 sec, the etch radius will vary from  $0.0009 \mu\text{m}$  to  $0.011 \mu\text{m}$ . The cell size must be smaller or equal to the smallest etch radius, so as to avoid discretization error. So, for a cell-size of  $0.0005 \mu\text{m}$  and a simulation area of  $2 \mu\text{m} \times 1 \mu\text{m}$ ,  $4000 \times 2000$  cells are required for the simulation! This eight-million-cell simulation will undoubtedly consume huge amounts of memory and computation time.

But what happens if the cell size is larger than  $0.0009 \mu\text{m}$ ? Suppose the cell size is  $0.01 \mu\text{m}$ , so that only  $200 \times 100$  cells are used in the whole  $2 \mu\text{m} \times 1 \mu\text{m}$  simulation area. Now, at the start of the simulation, the etch-rate at a boundary cell at coordinates  $(0,0,0)$  is





**Figure 4.6 :** Etch-rate vs depth in the photoresist. The data is taken from a vertical cut across the center of the etch-rate distribution plotted in Figure 2.3.



**Figure 4.7 :** Simulated resist profiles at a development time of 5 seconds, using the modified cell and string algorithms. The simulations use part of the etch-rate distribution plotted in Figure 2.3.

approximately  $0.03 \mu\text{m}/\text{sec}$ , from Figure 4.6. For a 0.1 sec time-step, the etch-radius of the cell is  $0.003 \mu\text{m}$ . The cell size is  $0.01 \mu\text{m}$ , so the etch-radius does not hit the center of the cell next to the boundary cell, and the boundary does not move. The etch area does not accumulate with time, so this sequence will be repeated for every subsequent time-step. Thus, in effect, because the cells are too large, the etch boundary does not move at all! Therefore, in order for the modified cell algorithm to work, the cell size has to be comparable in size or smaller than the smallest etch radius. Small time-steps have to be used for accuracy, and the etch rate typically has small values, so as a result, the resist development simulation will have to use a large number of cells. As a direct result, large amounts of memory and computation time are required for the simulation.

Figure 4.7 shows the profile of an isolated space in photoresist after 5 seconds of development, simulated with the modified cell algorithm. For comparison, the *SAMPLE* string simulation is overlaid. As in the resist simulation of Chapter 3, this simulation used the etch-rate distribution plotted in Figure 2.3. The modified cell simulation was run with a time-step of 0.1 seconds, and a cell size of  $0.00025 \mu\text{m}$ . The simulation was limited to a  $0.5 \mu\text{m} \times 1.0 \mu\text{m}$  area to save time; this corresponded to a total of  $2000 \times 4000$  cells. It is clear from Figure 4.7 that the profiles simulated using the two techniques do not agree. The discrepancy between the two profiles is most probably due to discretization errors in the cell algorithm. It is also important to note that the modified cell simulation lasted about an hour on a SUN 4/280, compared to some 10 seconds for the *SAMPLE* string simulation. Furthermore, if the cell size is increased to  $0.001 \mu\text{m}$ , the profile does not change, but remains in its initial straight-surface configuration.

#### 4.6. SUMMARY

The modified cell algorithm is easy to implement in both two and three dimensions. The algorithm, based on Huygens principle, is physically correct and robust. As a volumetric etching algorithm, no loops are formed during the simulation. In addition, the algorithm etch fronts produced by this algorithm are more spherical than those produced using the cell-removal algorithm.

There are two sources of error in this algorithm. The first comes from the discretization of circles with square cells. This discretization error can propagate and affect the later simulations. The discretization error may, however, be reduced by increasing the cell density so that more cells are swept or touched during each time-step. In order to retain accuracy, a typical Huygens etch should have a radius of approximately 10 cells. But this in turn means that the material being etched must contain a large number of cells to begin with.

The second type of discretization error occurs when the size of the cell is large compared to the radius of the Huygens etch. The algorithm does not keep track of the area of the cell that has been etched; it only knows whether the cell has been etched or not etched. Because of this, etching does not accumulate, and the etch front does not move. To handle this, the cell size has to be comparable in size or smaller than the smallest etch radius.

In order to accurately simulate the etching of photoresist, a large number of cells, perhaps on the order of  $10^3$  per micron, have to be used. In a 3D simulation of  $1\mu\text{m} \times 1\mu\text{m} \times 1\mu\text{m}$  cube, some  $10^9$  cells are required. With this number of cells, it is not difficult to see that the modified cell method will require lots of computer memory. Computation speed is another issue of considerable importance. In this algorithm, the computation speed is directly

proportional to the number of cells on the simulation boundary. The modified cell algorithm is similar to the Dill cell model in this respect. But the modified cell method uses a constant-size time-step, whereas the Dill method determines the time-step based on the fastest etch-speed of all the cells in the boundary. Therefore the modified cell method is more efficient and faster than the Dill model. Nevertheless, if the algorithm has to keep track of some  $10^6$  boundary cells in a volume filled with  $10^9$  cells, the algorithm undoubtedly will be quite slow.

It appears that the modified cell algorithm is useful for simple etch functions and might come in handy for simulating isotropic uniform etching. But at present, the algorithm cannot handle widely fluctuating etch-rates such as are found in photoresist development. The problems encountered with inhomogeneous etch-rates and discretization errors have interesting implications on other integer-based cell algorithms such as the spill-over technique (Section 3.5) as well. Nevertheless, this modified cell algorithm is quite promising, and should be explored further.

## REFERENCES

1. M. Born, E. Wolf, in *Principles of Optics, Sixth Edition*, Pergamon Press, London 1980.

## **CHAPTER 5**

### **THE MATHEMATICAL BASIS OF THE SURFACE-ADVANCEMENT ALGORITHMS**

#### **5.1. INTRODUCTION**

The string<sup>1</sup> and the ray<sup>2</sup> methods are both surface-etching algorithms that have been used successfully for calculating 2D profiles in etching simulations. These algorithms are more accurate, faster and require far less memory than any of the cell-etching methods discussed thus far. However, since both the string and the ray methods describe only the surface of the material being etched, each algorithm does have certain characteristic weaknesses.

Before launching into a detailed description of the string and the ray etching algorithms, it is most useful to examine the mathematical foundation of each of these algorithms. These two methods have traditionally been treated as separate algorithms, but it shall be shown in this chapter that both algorithms are actually based on the same mathematical solution to the general problem of tracing a time-evolving surface. And as shall be seen in later chapters, the differences in the way this mathematical solution is implemented lead to key differences in both the accuracy and the robustness of the simulations.

In this chapter, basic ways of viewing and solving the general problem of surface-advancement will be discussed. It shall be shown that a straight-forward solution cannot be obtained due to the implicit nature of the differential equation governing the surface-advancement. Instead, the principle of least-time must be used to determine the advancement of the etch surface as a function of time. The formal derivation based on the principle of least time leads to two complementary equations that link the string and the ray algorithms.

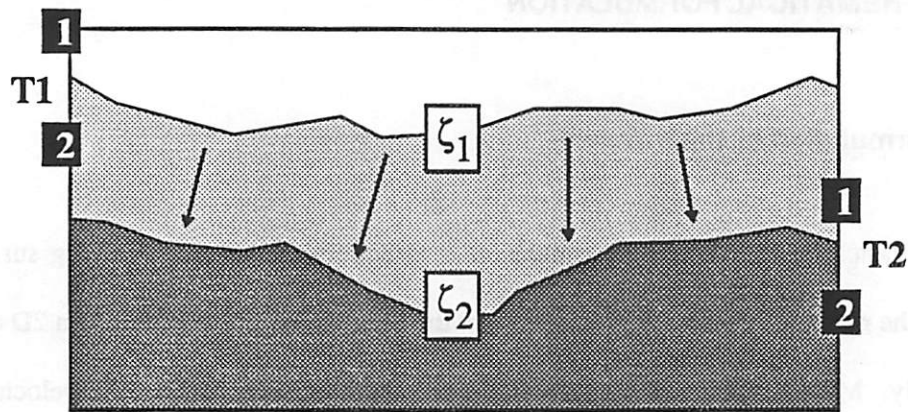
## 5.2. MATHEMATICAL FORMULATION

### 5.2.1. Formulation of the Problem

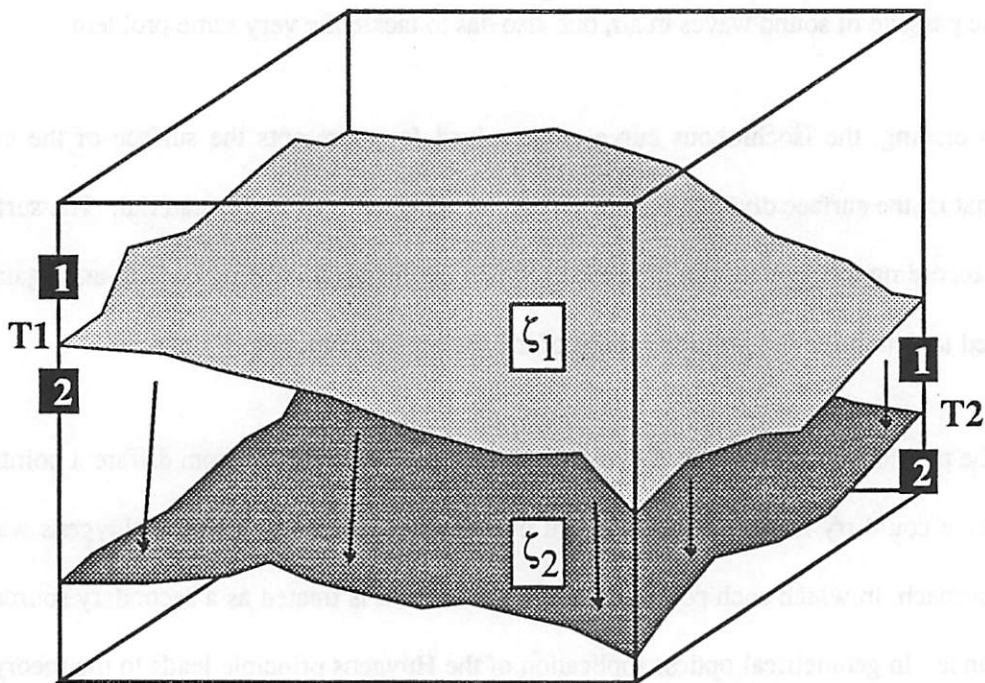
The basic goal of the etching simulation is to determine the time-evolving surface or profile of the material. Figures 5.1 and 5.2 show the formulation of the problem in 2D and 3D respectively. Mathematically, the problem can be stated as follows : Given a velocity field  $v(x,y,z)$  and a curve  $\zeta_1$  that divides regions 1 and 2 at time  $T_1$ , find at time  $T_2$  the shape and position of the curve  $\zeta_2$ , where  $\zeta_2$  is the new interface between regions 1 and 2. This very basic problem is encountered in many different branches of physics. In geometrical optics, for example, the curve could be the wavefront of light propagating through some arbitrary medium. To understand the behavior of waves rippling on the surface of water, or to determine the passage of sound waves in air, one also has to tackle the very same problem.

In etching, the isochronous curve to be solved for represents the surface of the etch-front, that is, the surface dividing the etched and unetched regions in the material. The surface moves according to the etch-rate or speed at which the material is being etched, and again, it is desired to determine the profile or shape of the surface as a function of time.

The problem of tracing a time-evolving surface can be examined from different points of view. One could try to trace the movement of the entire surface by using a Huygens wave-front approach, in which each point on the surface or front is treated as a secondary source of disturbance. In geometrical optics, application of the Huygens principle leads to the theory of diffraction, which is perhaps the basic postulate of the wave theory of light. And in etching, the Huygens principle could be used to trace out the etch-front by treating each point on the surface as an expanding etch wavelet. An implementation of the Huygens principle using cells



**Figure 5.1 :** 2D Etching : The surface advancement problem is that of tracking the 2D surface as it moves in time.



**Figure 5.2 :** 3D Etching : The surface movement problem is to determine the shape and location of the 3D surface as it moves in time.



was discussed in some detail in the previous chapter.

Another way to view the general problem is to divide up the curve into smaller segments or even points. Instead of tracing the movement of the entire curve, it is useful to consider the movement of a single point on the curve. The problem then is reduced to finding the path or trajectory of a point in space as it moves in time. This problem is in fact very closely related to the problem in geometrical optics of tracing a light ray as it passes through some optical media. If this trajectory or ray can be determined mathematically, then the curve can be reconstructed out of the end-positions of various initial points.

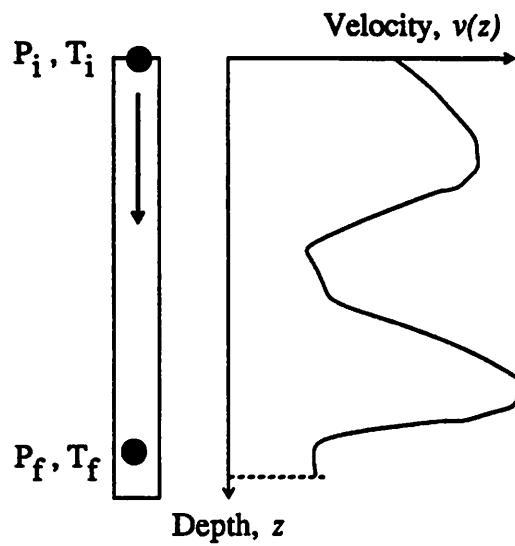
### 5.2.2. The Etching Problem In One-Dimensional Space

The "simplified" etching problem is to determine the location of a single point on the etch-front as it moves in time. The 1D problem may be stated as follows : Given that a particle is moving in a velocity field  $v(z)$ , where the velocity is a function of position, find the position of the particle at some time  $t$ . Unfortunately, when  $v(z)$  is position-dependent, a problem arises in that it is not possible to solve explicitly for the distance  $\Delta Z$  moved in some time  $\Delta T$ .

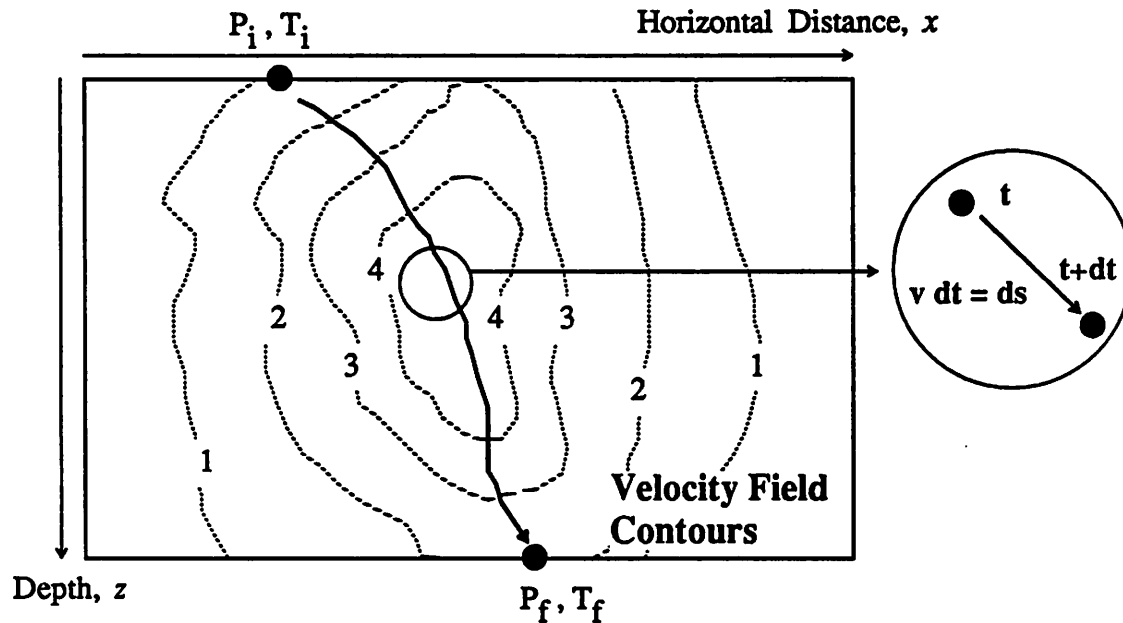
Figure 5.3 shows the one-dimensional movement of the particle. In some time interval  $dt$ , the particle travels a distance

$$v(z) dt = dz \quad [5.1]$$

The total distance traveled as a function of time is found from integrating the expression above. Note that the velocity is a function of position and not of time, so  $v(z)$  must be moved to the right side of the equation before the integration is performed. Thus, the integral becomes



**Figure 5.3 :** 1D Etching is equivalent to tracking the position or trajectory of a particle as it moves through a position-dependent velocity field.



**Figure 5.4 :** 2D and 3D Etching is equivalent to tracking the position or trajectory of a particle as it moves through a position-dependent velocity field, where the velocity is a function of the coordinates  $(x,y,z)$ . In some time  $dt$ , the particle moves a distance  $ds = v(x,y,z) dt$ .

$$\int_{T_i}^{T_f} dt = \int_{Z_i}^{Z_f} \frac{dz}{v(z)} \quad [5.2]$$

$$\Delta T = T_f - T_i = \int_{Z_i}^{Z_f} \frac{dz}{v(z)} \quad [5.3]$$

where  $Z_i$  is the location of the particle at some initial time  $T_i$ , and  $Z_f$  is the location at the final time  $T_f$ . The expression above can be evaluated if  $Z_i$  and  $Z_f$  are known or given; if  $T_i$  is also given, then the final etch-time  $T_f$  can be solved for by doing the integration and evaluating the limits.

The problem becomes more complicated when one tries to determine how far the particle moves in some time-step  $\Delta T$ . Equation [5.3] states that the time-interval  $\Delta T$  is a function of the total distance traversed  $\Delta Z$ , i.e.,

$$\Delta T = f(\Delta Z) \quad [5.4]$$

Note that Equation [5.3] is *not* an explicit function of  $\Delta T$ . Thus, given some time-step  $\Delta T$ , a straight-forward evaluation of [5.3] for  $\Delta Z$  or  $Z_f$  is not possible. In theory, to obtain  $\Delta Z$ , one could determine the inverse  $f^{-1}$  of the function  $f$  such that

$$\Delta Z = f^{-1}(\Delta T) \quad [5.5]$$

But the function  $f$  is often not a simple function, as it is itself the integral of the inverse of a velocity function. Therefore the inverse function  $f^{-1}$  might not exist. So, in general, it is not possible to evaluate  $\Delta Z$  explicitly as a function of the time-step  $\Delta T$ . The only recourse is to evaluate Equation [5.3] numerically to determine the distance traversed by the particle as a function of time.

Clearly, it is not easy to find the distance traveled by a particle during some time-step  $\Delta T$  in a position-dependent velocity field, even when the problem is restricted to one dimension. A straight-forward integration does not work, because the velocity field is not a function of time. It is possible to solve the problem numerically, but the result will be sensitive to the

choice of numerical algorithm and to the internal details of the numerical algorithm itself. Unfortunately, the problem does not get any easier in 3D.

### 5.2.3. The Etching Problem in Three-Dimensional Space

In 3D, the etching problem can be formulated as follows : Given that a particle is moving in a velocity field  $v(x, y, z)$ , find the position or trajectory of the particle as it travels in time. Figure 5.4 shows the movement of a particle in a two-dimensional velocity field. Similarly, if the velocity field is extended to three dimensions, the particle will be able to move in three dimensions. From practical experience, it is known that the trajectory of the particle will be affected by the local velocity field. If the field is a fast-moving stream of water, and the particle a floating leaf, the leaf might be propelled towards the center of the stream where the water flows fastest, or it might, as often happens, be pushed towards the shore. What controls the movement of this leaf? Where will the leaf go? Or more generally, is it possible to determine where the particle will move to, given knowledge of the velocity field and the particle's initial conditions?

As in the 1D case, it is possible to write a differential equation for the distance traveled by the particle in some time  $dt$ . If  $ds$  is the change in the position of the particle, then

$$ds = v(x, y, z)dt \quad [5.6]$$

The position  $s$  and velocity  $v$  of the particle are functions of the coordinates  $(x, y, z)$ , so the differential equation can be integrated over the path of the particle as it moves from point  $P_i$  to  $P_f$  in some time  $\Delta T = T_f - T_i$ .

$$\int_{T_i}^{T_f} dt = \int_{P_i}^{P_f} \frac{1}{v(x, y, z)} ds \quad [5.7]$$

This situation is illustrated in the inset of Figure 5.4. The problem with Equation [5.7] is that

the velocity is a scalar function; [5.7] tells us how *far* the particle moves, but not the *direction* in which it moves. There is no information in either [5.6] or [5.7] as to changes in direction due to the changing velocity field. Another problem is that the integral in [5.7] has to be evaluated over the path or trajectory  $s$  of the particle. But this path is not known to begin with! Thus, it is not possible to evaluate [5.7]. And for the same reasons, it is pointless to discretize Equation [5.6] or [5.7] to find the trajectory or distance traveled by the particle.

#### 5.2.4. The Principle of Least Time

The problem of determining the trajectory of the particle can be solved if it is assumed that the path along which the particle travels minimizes the transit time. There are many curves joining the points  $P_i$  and  $P_f$ , but there is only one curve on which the particle's travel time is minimized. This assumption is actually a statement of the principle of least time.

The principle of least time, also known as *Fermat's Principle*, asserts that the time

$$T = \int_{T_i}^{T_f} dt = \int_{P_i}^{P_f} \frac{1}{v(x,y,z)} ds \quad [5.8]$$

for a particle to move along an actual ray between two points  $P_i$  and  $P_f$  is shorter than the time taken along the path of any other curve which joins these points.† The problem of finding this minimum time can be solved using variational calculus. If the time  $T$  is to be a minima over the true path, then if the path of integration is changed slightly, there must be, to first order, no change in  $T$ . The variation in  $T$ ,  $\delta T$ , must be zero. This is stated as follows.

---

† *Fermat's Principle* actually refers to optical path lengths in geometrical optics. However, since

$$\int \frac{c}{v} ds = c \int dt$$

the principle of the shortest optical path is also the principle of least time. For proof of *Fermat's Principle*, see Born & Wolf,<sup>3</sup> Section 3.3.2.

$$\delta T = \delta \int_{P_i}^{P_f} \frac{1}{v(x,y,z)} ds = 0 \quad [5.9]$$

Using this as a beginning point, it is possible to derive a differential equation describing the trajectory of a ray. (A derivation using variational calculus is shown in Appendix A.1.) If  $\mathbf{r}$  is a position vector of a typical point on a ray, and  $s$  the length of the ray measured from a fixed point on it, then

$$\frac{d}{ds} \left( \frac{1}{v(x,y,z)} \frac{d\mathbf{r}}{ds} \right) = \nabla \left( \frac{1}{v(x,y,z)} \right) \quad [5.10]$$

This equation, typically referred to as the *differential ray equation*, is the basic equation describing the trajectory of a ray in some velocity field  $v(x,y,z)$ . The unit vector  $\mathbf{s}$  is related to the position vector  $\mathbf{r}$  by

$$\mathbf{s} = \frac{d\mathbf{r}}{ds} \quad [5.11]$$

As a consequence, the differential ray equation may be written as

$$\frac{d}{ds} \left( \frac{1}{v(x,y,z)} \mathbf{s} \right) = \nabla \left( \frac{1}{v(x,y,z)} \right) \quad [5.12]$$

Figure 5.5 describes the behavior of a ray and its relationship to the differential ray equation.

As a ray moves through space, it will be deflected by the local velocity field. The change in the direction  $\mathbf{s}$  of the ray is related to the velocity field by the differential ray equation.

Related to the differential ray equation is the *eikonal* equation.

$$|\nabla \zeta|^2 = n^2 \quad [5.13]$$

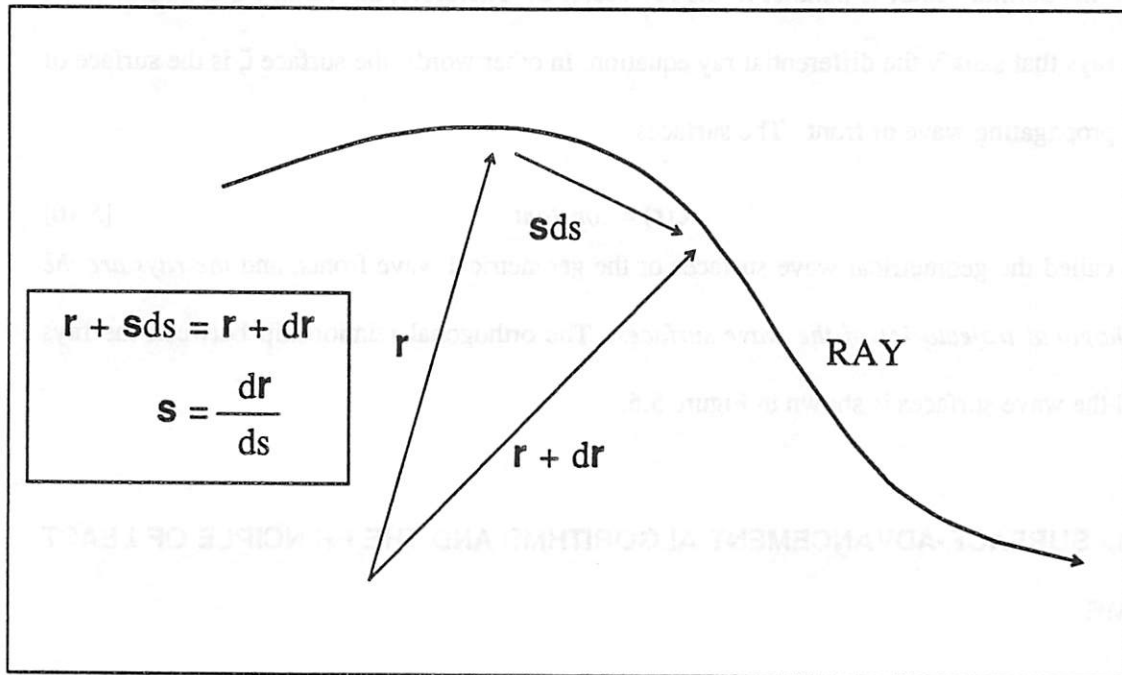
This may be written explicitly as

$$\left[ \frac{\partial \zeta}{\partial x} \right]^2 + \left[ \frac{\partial \zeta}{\partial y} \right]^2 + \left[ \frac{\partial \zeta}{\partial z} \right]^2 = n^2(x,y,z) \quad [5.14]$$

The eikonal can also be written in terms of the unit vector  $\mathbf{s}$ , where  $\mathbf{s}$  is, as above, the unit vector in the direction of the ray.

## THE RAY EQUATION

$$\frac{d}{ds} \left( \frac{1}{v(\mathbf{r})} \mathbf{s} \right) = \nabla \left( \frac{1}{v(\mathbf{r})} \right)$$



### DEFINITIONS :

$s$  : length of the ray

$ds$  : incremental line-element

$\mathbf{s}$  : unit vector in the direction of the ray

$\mathbf{r}$  : position vector

$v(\mathbf{r})$  : velocity-field as function of position

**Figure 5.5 :** The Ray Equation : The change in the direction of a ray is proportional to the gradient of the inverse of the velocity field.

$$\nabla\zeta = n \mathbf{s} \quad [5.15]$$

It is shown in Appendix A.2 that the eikonal equation satisfies the differential ray equation, which in turn, is a solution to the least time principle. The ray and eikonal equations can be understood as follows. Suppose there is a surface  $\zeta(x, y, z) = \text{constant}$ . By definition, the gradient of the surface  $\zeta(x, y, z)$  is the normal vector to this surface. But the eikonal [5.15] states that this normal vector is parallel to the ray vector  $\mathbf{s}$ . Therefore, the surface  $\zeta$  is normal to all the rays that satisfy the differential ray equation. In other words, the surface  $\zeta$  is the surface of the propagating wave or front. The surfaces

$$\zeta(\mathbf{r}) = \text{constant} \quad [5.16]$$

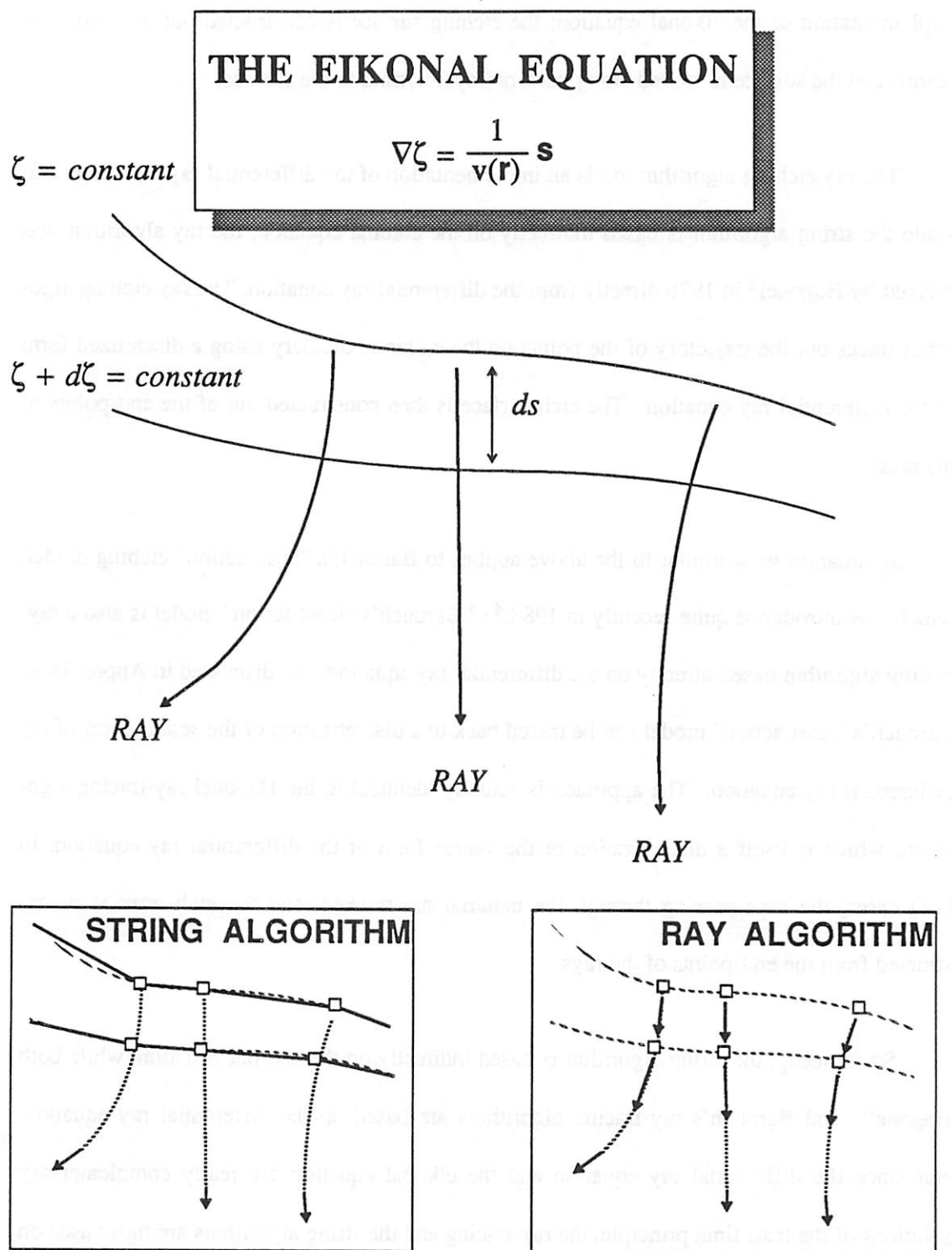
are called the geometrical wave surfaces or the geometrical wave fronts, and *the rays are the orthogonal trajectories of the wave surfaces*. The orthogonal relationship between the rays and the wave surfaces is shown in Figure 5.6.

### 5.3. SURFACE-ADVANCEMENT ALGORITHMS AND THE PRINCIPLE OF LEAST TIME

The eikonal and the differential ray equations are the two basic equations applicable to the simulation of etching. In fact, the string algorithm for etching is based on the eikonal equation, while the ray etching algorithm is derived from the differential ray equation.

The string etching algorithm was originally conceived by Jewett<sup>1</sup> in 1977 as a "common-sense" method for discretizing the movement of the etch-boundary in time. In the string model, the etching boundary between etched and unetched regions is approximated by a series of points joined by straight line segments. Jewett<sup>1</sup> states that "each point advances along the angle bisector of the two adjoining segments...". But notice that the angle bisector is also the normal vector to the local surface. Therefore the string algorithm is actually an





**Figure 5.6 :** The rays are the orthogonal trajectories of the wave surfaces  $\zeta(x,y,z) = \text{constant}$ ,  $\zeta$  being a solution of the eikonal equation. The string algorithm is based on the eikonal, while the ray algorithm is based on the ray equation.

implementation of the eikonal equation; the etching surface is constructed out of a string of points, and the surface is moved along vectors (rays) normal to the surface.

The ray etching algorithm too is an implementation of the differential ray equation. But while the string algorithm is based indirectly on the eikonal equation, the ray algorithm was derived by Hagouel<sup>2</sup> in 1976 directly from the differential ray equation. The ray etching algorithm traces out the trajectory of the points on the etching boundary using a discretized form of the differential ray equation. The etch surface is then constructed out of the end-points of the rays.

A situation very similar to the above applies to Barouch's "least action" etching model, which was introduced quite recently in 1988.<sup>4, 5</sup> Barouch's "least action" model is also a ray-tracing algorithm based directly on the differential ray equation. As discussed in Appendix A, Barouch's "least action" model can be traced back to a discretization of the *scalar* form of the differential ray equation. The approach is actually identical to the Hagouel ray-tracing algorithm, which is itself a discretization of the *vector* form of the differential ray equation. In both cases, the rays passing through the material are tracked, and the etch-front is reconstructed from the end-points of the rays.

So, to recap, the string algorithm is based indirectly on the eikonal equation, while both Hagouel's and Barouch's ray tracing algorithms are based on the differential ray equation. But since the differential ray equation and the eikonal equation are really complementary solutions of the least time principle, the ray-tracing and the string algorithms are thus based on the same principles.

In practice, however, the calculation of the advancement vector based on the ray equa-

tion or the surface normal does affect the simulated profiles. As shall be seen in the next chapter, the method of calculating the vector turns out to be a very important factor distinguishing the ray algorithm from the string etching algorithm.

## REFERENCES

1. R.E. Jewett, P.I. Hagouel, A.R. Neureuther, T. Van Duzer, "Line-Profile Resist Development Simulation Techniques," *Polymer Eng. Sci.*, vol. 17, no. 6, pp. 381-384, June 1977.
2. P.I. Hagouel, "X-ray Lithographic Fabrication of Blazed Diffraction Gratings," *Ph.D. Dissertation*, University of California, Berkeley, 1976.
3. M. Born, E. Wolf, in *Principles of Optics, Sixth Edition*, Pergamon Press, London 1980.
4. E. Barouch, B. Bradie, S.V. Babu, "Calculation of Developed Resist Profiles by Least Action Principle," *Interface'88 : Proceedings of KTI Microelectronics Seminar*, pp. 187-196, November 1988. 2D Ray method.
5. E. Barouch, B. Bradie, H. Fowler, S.V. Babu, "Three-Dimensional Modeling of Optical Lithography for Positive Photoresists," *Interface'89 : Proceedings of KTI Microelectronics Seminar*, pp. 123-136, November 1989. 3D Ray method.

## CHAPTER 6

### THE STRING AND THE RAY

### SURFACE-ADVANCEMENT ALGORITHMS

#### 6.1. INTRODUCTION

As discussed in the previous chapter, the string and the ray etching algorithms are based on the very same mathematical solution to the general problem of tracing a surface or profile as it evolves in time. But as shall be seen in this chapter, the string and the ray algorithms do differ significantly in the accuracy, robustness and also completeness of the simulation. These differences are directly related to the implementation of the above-mentioned "basic" mathematical solution.

In this chapter, the two algorithms shall be discussed in some detail. Since both of these algorithms are really discrete methods of solving for the approximate etch-surface, emphasis will be laid upon determining those techniques and modifications required for improving the accuracy, correctness, efficiency and robustness of the simulations.†

#### 6.2. THE STRING ALGORITHM

The 2D string model, proposed by Jewett,<sup>1</sup> uses a "string" of points or nodes to approximate the etching boundary between the etched and unetched regions. Each point or node advances along a vector normal to the local surface. This advancement or direction vector is, as shown in Figure 6.1, the average of the normal vectors of the segments adjacent to the

---

† Accuracy and correctness are used in this chapter to refer to different aspects of the simulations. Accuracy refers to the conformity of the result to the "true" value, while correctness is used to refer primarily to the formation of loops.

node. As mentioned before, this is exactly the condition of the eikonal; the rays that satisfy the least time principle are also normal to the etch front.

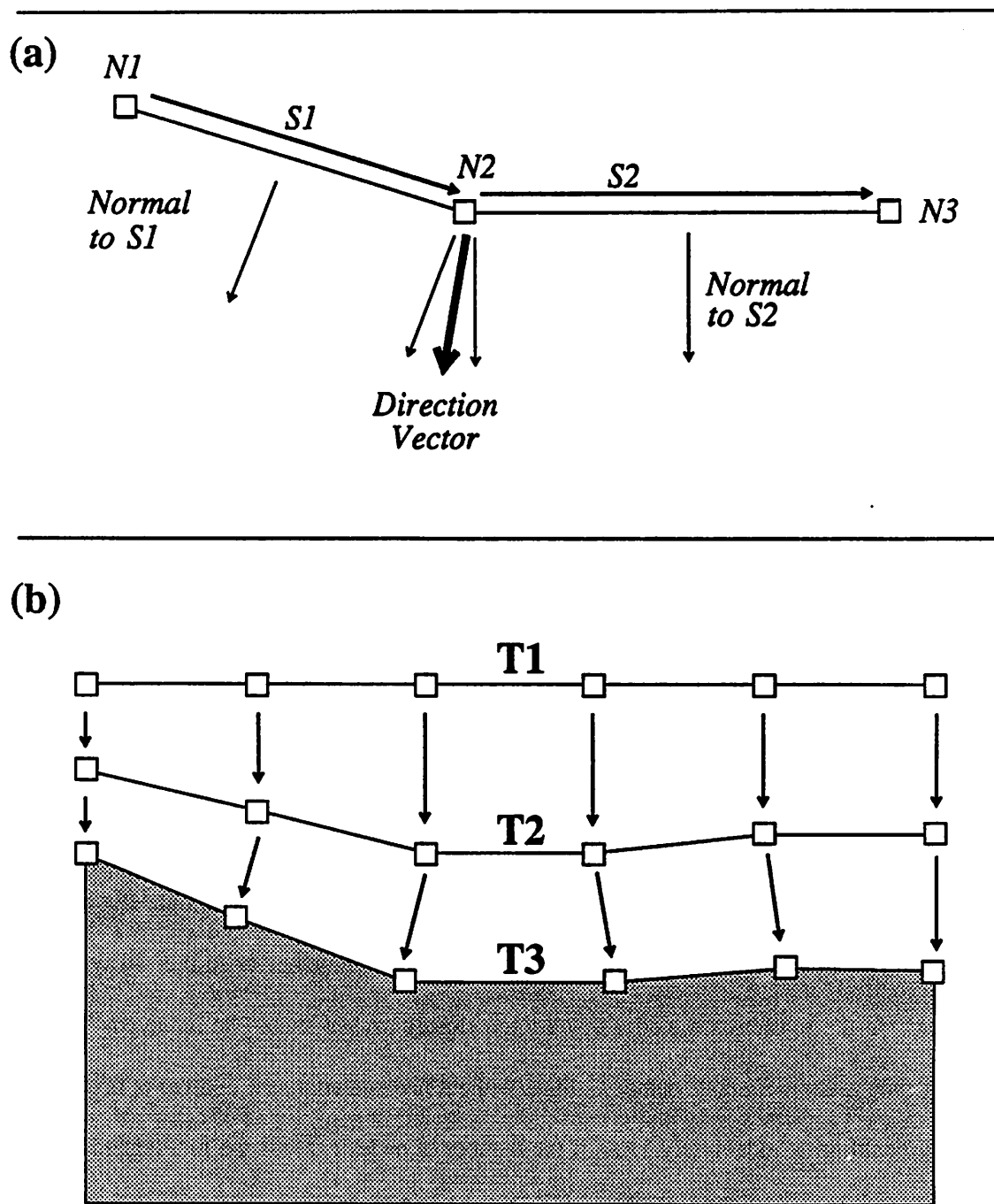
### 6.2.1. The Algorithm for String Advancement

A typical string, consisting of 50 - 100 segments, is started on the surface of the material. The algorithm for moving the string is as follows :

- [I] For each node on the string, find the node's direction vector by averaging the normals to the adjoining segments.
- [II] Advance each node along its direction vector. The distance advanced is equal to the local etch-rate multiplied by the (constant) incremental time-step.
- [III] Add nodes where the string has expanded, and delete nodes in regions of contraction.
- [IV] Proceed to [I] and repeat until the total etch-time has been reached.

### 6.2.2. Implementation of the String Algorithm

The string method is not particularly easy to set up in 2D, because of the logistical problem of keeping track of all the nodes and segments that make up the etching boundary. The problem is complicated by the need to keep track of the ordering of the nodes on a segment. The need for order in turn is imposed by the method of advancing the surface. For a node to find its direction or advancement vector, it needs to know the normal vectors of its adjoining segments. And to find the normal vector of a segment, the segment must know the coordinates of the two nodes that make up the segment, as well as the *order* in which they occur. In other words, it is necessary to have a directed line segment in the form  $S = [N_1, N_2]$ , where  $N_1$  and



**Figure 6.1 :** The string model. (a) A node is advanced along a vector that is the average of the normal vectors of the adjacent segments. (b) The string is started on the surface of the material. Each node is moved by an amount  $ds = (\text{local etch-rate}) \times (\text{time-step})$ .

$N_2$  are the two nodes on the segment. This last condition comes about because the coordinates alone are not sufficient to define the direction of the vector normal to the segment; without any order to the nodes, the direction vector could point in either of two opposing directions.

In order to test the accuracy and determine the requirements or limitations of the string algorithm, the string algorithm has been implemented in the C programming language. The implementation is based on the FORTRAN string implementation in *SAMPLE*. As in *SAMPLE*, the string is implemented as an array of nodes. The segments are determined by the arrangement of the nodes within the array; the  $i$ -th segment thus contains the nodes at the  $i$  and  $i+1$  positions in the array, i.e.,  $S_i = [N_i, N_{i+1}]$ . This arrangement provides a natural order to the nodes on the string, and is equivalent to having an array of directed line segments.

### 6.2.3. Mesh Modification

The string algorithm uses an ordered list of nodes and segments to approximate the etch boundary separating the etched regions from the unetched regions in the material. But as the string moves in time, nodes on the string could move closer together or further away from each other. In the former case, nodes that are too closely packed should be removed. There are two strong reasons for node deletion. First, without node deletion, the number of nodes on the string could grow indefinitely. And if the number of nodes exceeds the pre-defined array size, the string algorithm will fail.† The second incentive for removing dense nodes is to avoid floating-point errors. If the nodes are packed too closely together, floating-point errors could

---

† The array implementation imposes a compile-time limit on the number of nodes in the string. Once the number of nodes exceeds the array size the program will fail to execute correctly. This problem could be avoided using a linked list implementation of segments and nodes. The linked list allows dynamic allocation of memory, so that the number of segments or nodes is limited only by the computer's memory size. Even so, it is still undesirable to allow the number of nodes to grow unchecked, since the computation time increases with the number of nodes on the string.

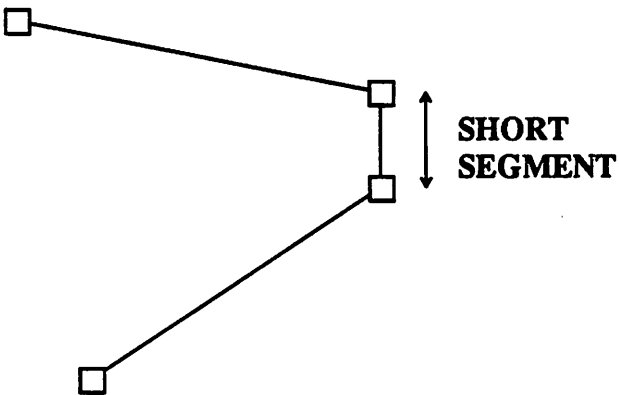
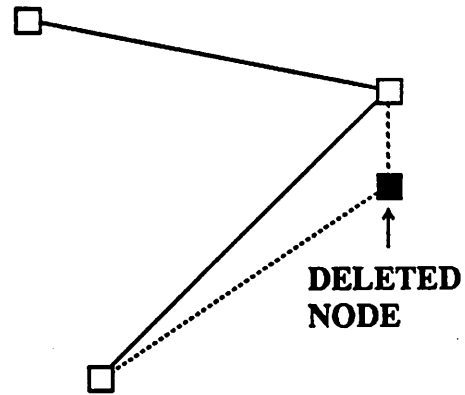


affect the relative positions of the nodes. The movement of the nodes depends very much on the ordering of the nodes, so if floating-point errors do occur, very small segments could become misoriented. As a result, the nodes would move in incorrect directions and the simulation itself could become incorrect.

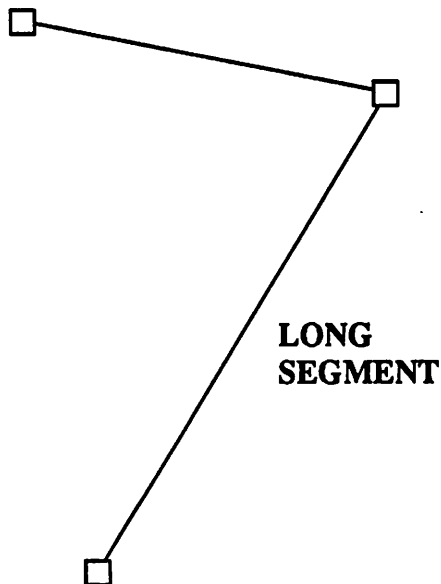
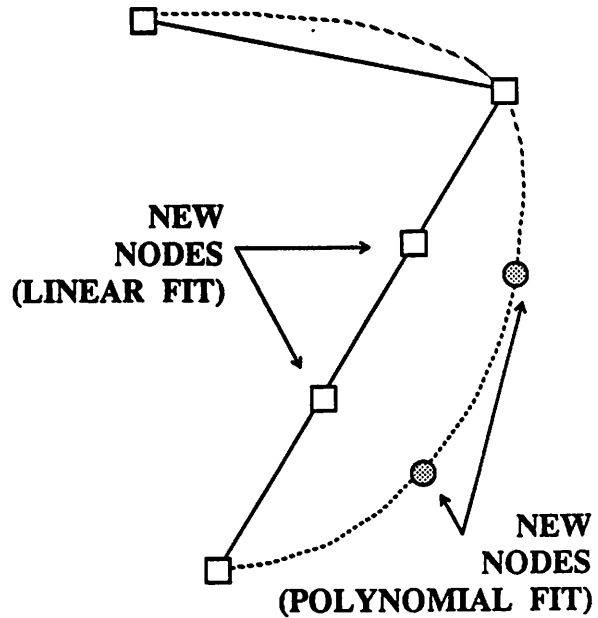
The reasons for node addition are somewhat different. Here, the primary concern is for accuracy. If the nodes on the string become too distant, the string will no longer be a good approximation to the true etched surface. So, nodes need to be added periodically so that the density of the nodes on the string remain approximately constant.

In *SAMPLE*, the nodes are added or deleted according to the length of the segment joining two nodes. A nominal or ideal segment size  $S_{ideal}$  is first defined, and minimum and maximum allowable segment lengths are defined in terms of this ideal segment length. After every surface advancement, the lengths of the segments are checked. If the segment is too short, i.e.,  $S < S_{min}$ , then one of the nodes on the short segment is removed, as shown in Figure 6.2. Or if the segment is too long, such that  $S > S_{max}$  as in Figure 6.3, then nodes need to be added to the string. The easiest way to do this is to add nodes linearly between sparse nodes. Alternately, a polynomial fit can be used to produce a smoother and perhaps more accurate surface. This is shown in the picture on the left in Figure 6.3.

As might be expected, the accuracy of the simulation does depend on the density of the mesh. If the etch-rate varies rapidly with distance, then shorter segments will produce more accurate results. Figure 6.4 shows the results of a circular uniform etch using the string algorithm. Figure 6.4a is a result of a simulation with a large ideal segment length  $S_{ideal} = 0.1 \mu\text{m}$ . The profiles at the earlier time-steps are clearly not very circular, but as the string expands, more nodes are added. After 10 time-steps, the string profile resembles the expected circular

**BEFORE DELETION****AFTER DELETION**

**Figure 6.2 :** Node Deletion : If a segment is shorter than some predefined minimum length, the segment and one of the nodes on the segment is deleted.

**BEFORE ADDITION****AFTER ADDITION**

**Figure 6.3 :** Node Addition : Nodes must be added to sparse regions of the string in order to preserve accuracy. Nodes can be added to long segments, using either a linear or polynomial fit.

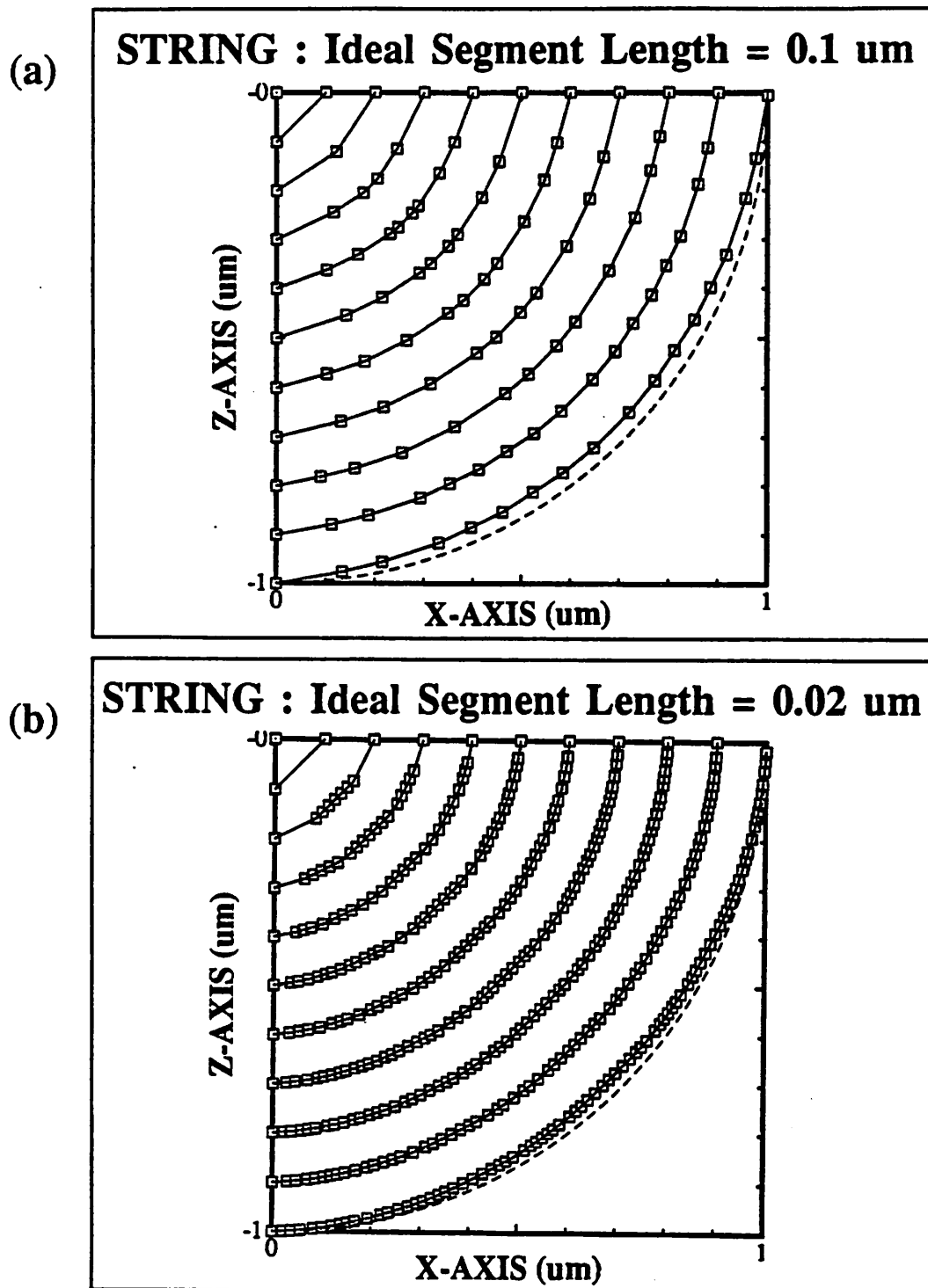


Figure 6.4 : 2D uniform etching beginning from a seed point at coordinates (0,0). The solid lines represent the string-etched profiles at etching times of 0.1 - 1.0 seconds. The expected result, a semi-circle, is plotted in a dashed line. The simulations were run with (a) large 0.1  $\mu\text{m}$  segments, and (b) small 0.02  $\mu\text{m}$  segments. The etch rate is 1  $\mu\text{m}/\text{sec}$ .

result. When a smaller ideal segment length  $S_{ideal} = 0.02 \mu\text{m}$  is used in the simulation, the profiles appear more circular even at the early stages of the simulation. However, more nodes are used in the simulation, and as a result, the computation time increases. Nevertheless, it is interesting to note that the computation time for the string simulation is, in this case, no more than a few seconds on a SUN 4/280. The string algorithm is much faster than either of the cell methods discussed in the previous chapters.

### 6.2.3.1. Rounding Errors at Diverging Corners

Unfortunately, the string modification process described above will not work accurately for sharp corners. The example shown in Figure 6.5 shows a string expanding outwards from a  $90^\circ$  diverging corner. The simulation begins with a grooved surface, and the profile etches outward with a uniform etch-rate. According to the Huygens principle of overlapping etch-fronts, the etch surface should be circular directly underneath the sharp corner, and linear far from the corner. † In Figure 6.5a, the simulation resulted in a significantly tapered profile even after 5 time-steps. The simulated profiles of Figure 6.5b, on the other hand, were run with the same segment size but with multiple nodes added at the corners. The resultant curves are quite satisfactorily circular. This result is due to the fact that the addition of multiple nodes at the sharp corners increases the accuracy of the simulation; the profile after the first time-step is a better approximation to the theoretical circular front. If no additional nodes are added, as in the simulation of Figure 6.5a, the circular front will be approximated by only two straight-line segments. As a result, as shown in the schematic of Figure 6.5a, rounding errors are introduced in the vicinity of the corner. The rounding errors propagate with the etch simulation and will affect the final profile.

---

† The Huygens principle is described in some detail in Chapter 4, Section 4.2.

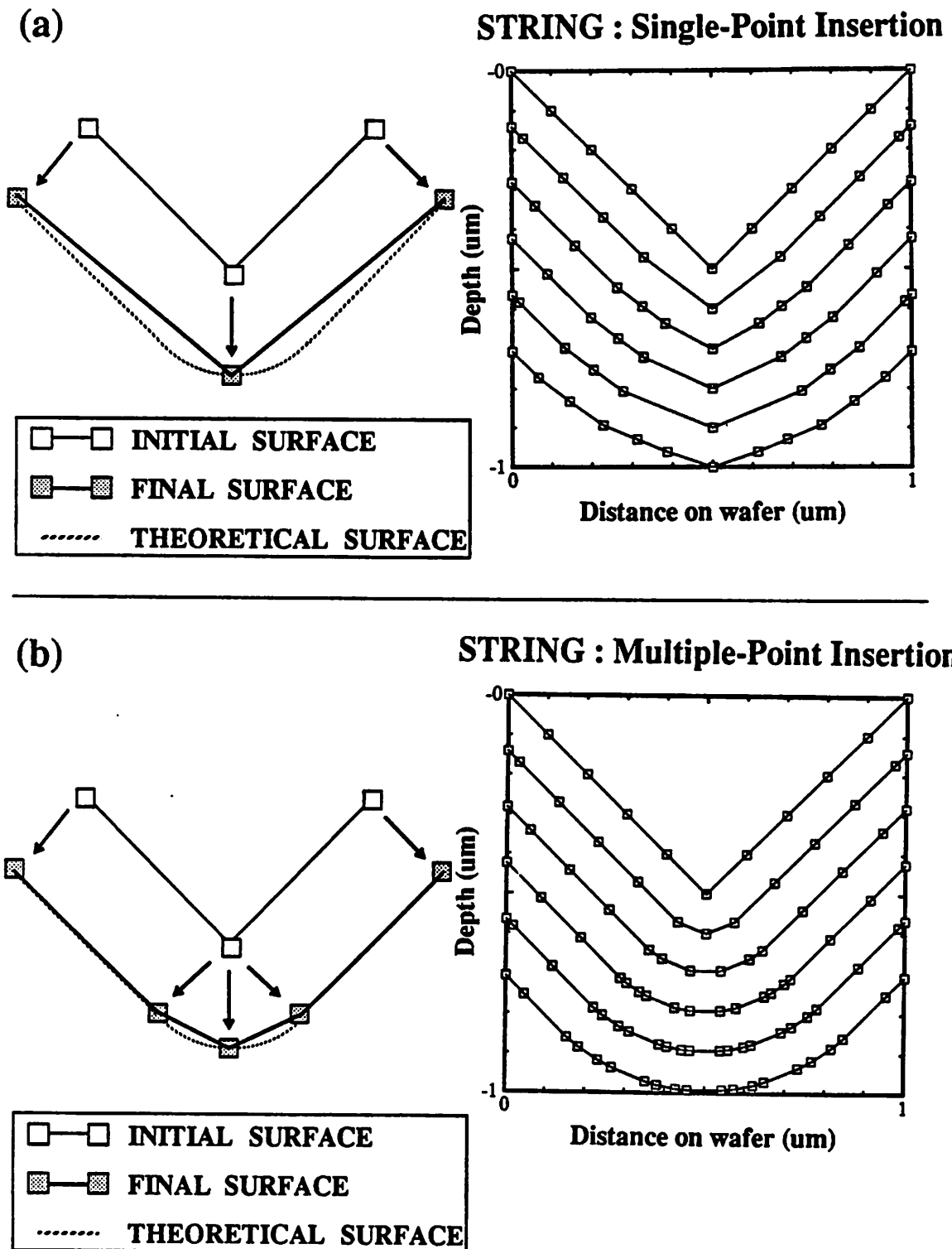


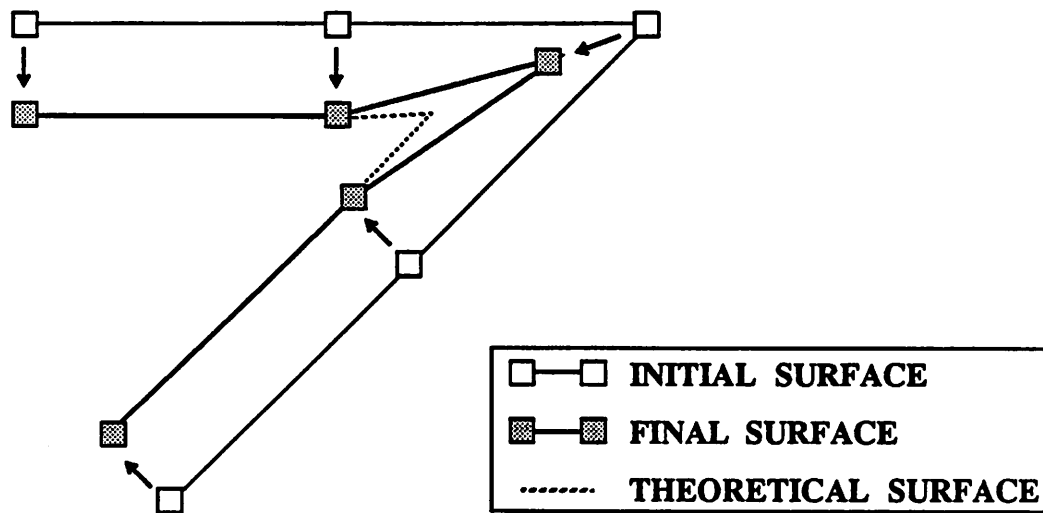
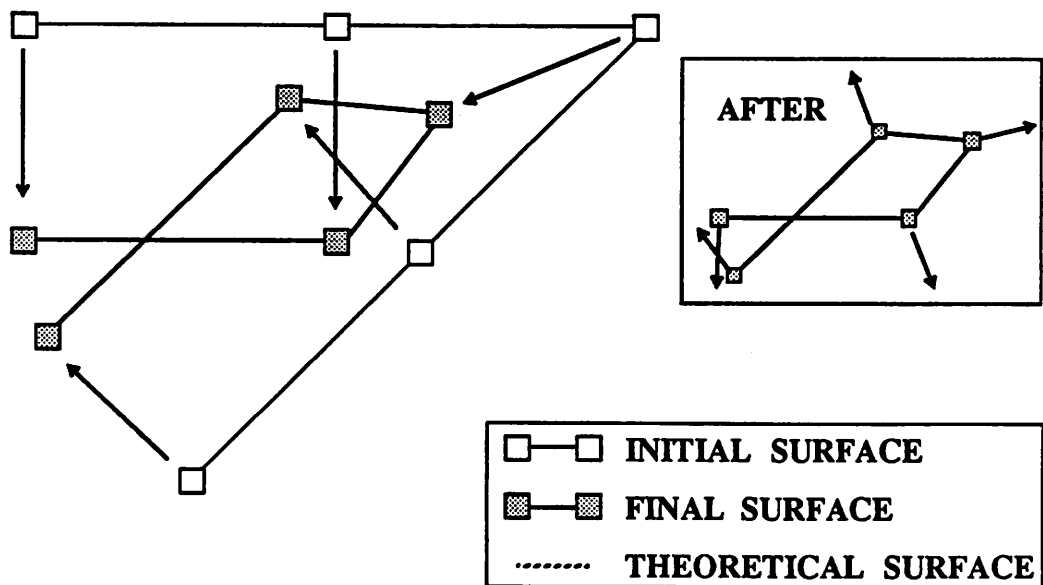
Figure 6.5 : (a) Rounding errors are introduced when only a single node is used to advance a sharp corner. (b) In order to follow more closely the theoretical or Huygen's etch front, more nodes should be added at the corners.

As a rule of thumb, multiple nodes should be added to the string if the angle between the two diverging segments at the corner is smaller or equal to  $90^\circ$ . In the example shown in Figure 6.5b, one node is placed at the bisector of the corner, while two other nodes are placed normal to the edges of the original segments.

It should also be pointed out that the sharp tapered profile in Figure 6.5a is partially due to the relatively large segments sizes ( $S_{ideal} = 0.2 \mu\text{m}$ ) used in this set of simulations. If the ideal segment size is kept relatively small, the accuracy of the profile will undoubtedly increase. In this particular diverging corner example, the rounding errors will become less significant if smaller segment sizes are used. But the key is that for a given segment size, greater accuracy will be obtained if multiple nodes are added to sharp diverging corners.

### 6.2.3.2. Scissoring and Looping at Converging Corners

The string algorithm also runs into difficulties at converging corners. "Scissoring" or "looping" are effects that take place at sharp converging corners. They represent instabilities and require corrective measures. Both effects occur when the etch front consists of a number of nodes on a sharp converging corner, as shown in Figure 6.6. In a physical sense, the corner is the intersection of two etch fronts. So, if the etch rate is uniform, both etch fronts will move along uniformly and the shape of the corner should remain unchanged. However, in the string algorithm, a node at the corner will move along a vector normal to the corner, while the nodes away from the corner will move normal to the straight sides of the corner (Figure 6.6a). As a result, a node at the corner does not move as far normal to the sides of the corner compared to the nodes far from the corner, even though each node might move the same distance. This causes a "scissoring" effect where the corner is effectively sharpened by the algorithm. In order to correct for this effect, it is necessary to increase the effective etch distance of the

**(a) SCISSORING****(b) LOOPING**

**Figure 6.6 :** Problems with the string algorithm. (a) Scissoring : Lagging nodes in converging corners cause inaccurate sharpening of the corners. (b) Looping : Segments can cross each other, forming loops.

corner node.

In a sharp corner, the segments could also move far enough that they intersect each other and form loops. This effect, shown in Figure 6.6b, can occur at converging corners, or at the intersection of converging etch fronts. The formation of loops is purely a result of the algorithm; it occurs because the surface is moved without any knowledge of whether one part of the surface is going to cross into an already etched region. String looping is a very serious problem because of its destructiveness. The string algorithm moves the etch front normal to the stringed surface. But when the surface is incorrect, as shown in the inset of Figure 6.6b, the algorithm will also produce incorrect results. In the example of Figure 6.6b, the loop will expand outwards, thus etching into already-etched regions. Unless the loop is removed, it will destabilize the entire simulation and cause the program to crash.

### 6.2.3.3. Loop Formation In Photoresist Development

Figure 6.7 shows the string-simulated profiles of photoresist development with the etch-rate distribution of Figure 2.3. After 15 seconds of development time (150 time-steps) small loops have formed at the standing wave corners. After 20 seconds (200 time-steps), the loops have increased in size and complexity, and have begun to intersect each other. 40 time-steps later, the loops are completely out of control; the profile now looks like a ball of yarn. There are more nodes in the loop than in the correct outside profile. Also, the number of nodes in the string has increased due to the loops. The program will fail a few time-steps later when the number of nodes increases beyond the pre-defined storage limit (typically 1000 nodes in *SAMPLE*).



### STRING : 15 sec Development Time

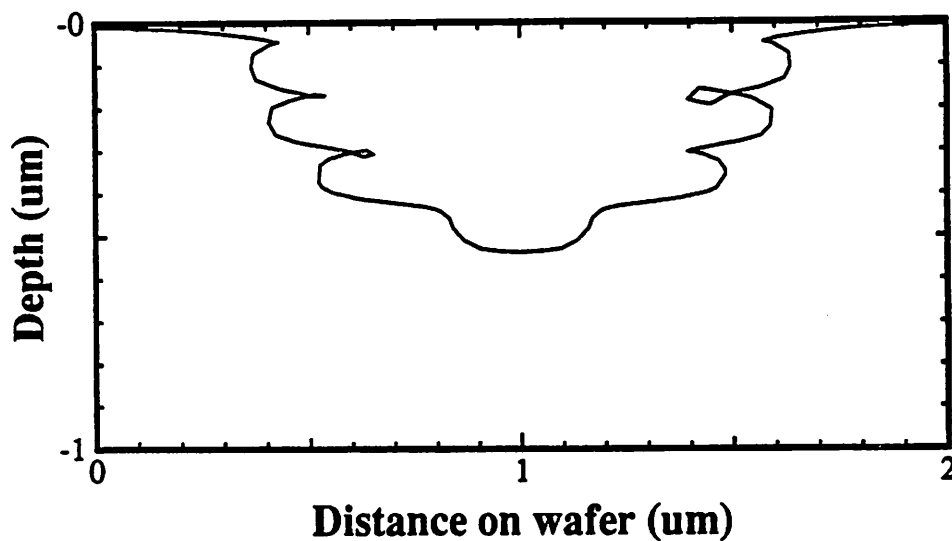


Figure 6.7a : String-simulated resist profile at development time of 15 seconds. The etch-rate distribution of Figure 2.3 is used. Loops have begun to form in the surface. The ideal segment length is 0.04 um.

### STRING : 20 sec Development Time

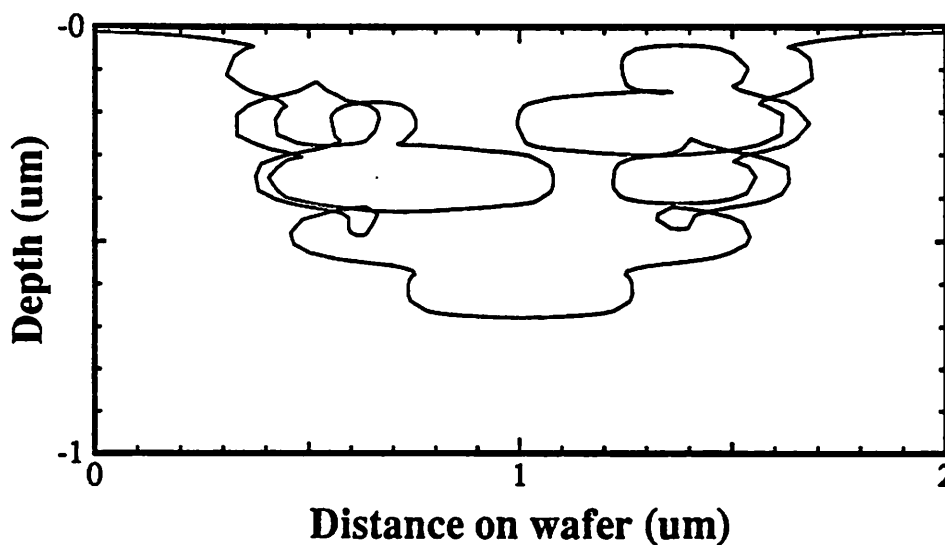


Figure 6.7b : String-simulated resist profile at development time of 20 seconds. The etch-rate distribution of Figure 2.3 is used. The loops have increased in size and complexity.

### STRING : 24 sec Development Time

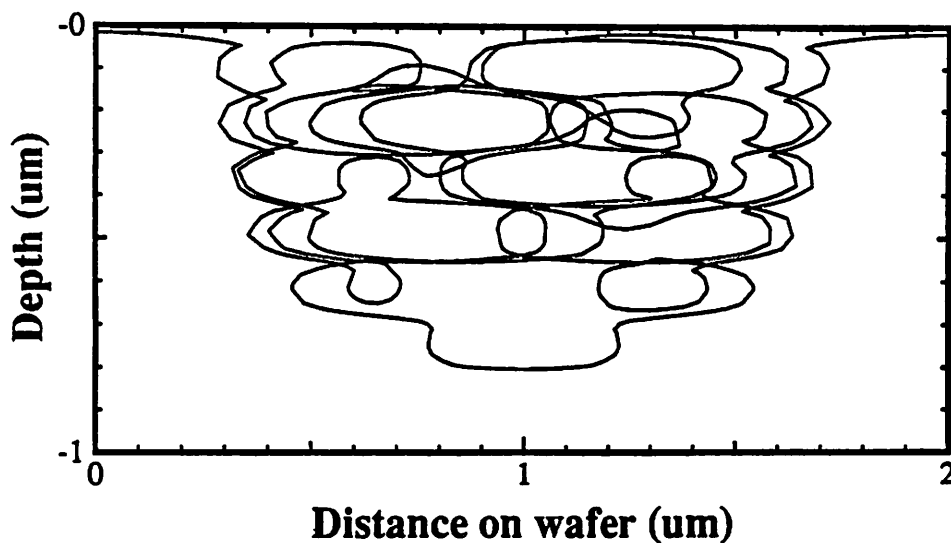


Figure 6.7c : String-simulated resist profile at development time of 24 seconds. The loops have blown up. The number of points in the string is close to the array size; the simulation will crash a few time-steps later.

### SAMPLE : 15, 20, 25 sec Development Time

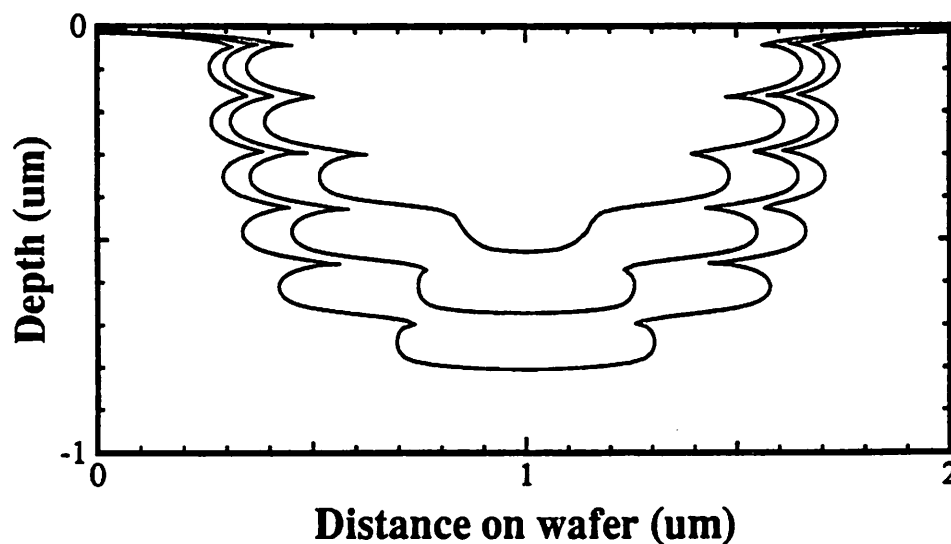


Figure 6.7d : Resist profile, simulated with *SAMPLE*, at development times of 15, 20 and 25 seconds. The loops have been removed by delooping at selected intervals.

It is very important to realize that the destructive loop behavior observed in this set of simulations is caused by the surface-advancement method itself.† The string algorithm moves the surface using vectors determined from the local surface. But if the local surface is itself incorrect, then clearly the advanced surface will also be incorrect. Another complication arises from the etch-rate distribution itself. In typical photoresist development, there are areas of low etch-rate sandwiched in between areas of high etch-rate. The loops are formed at the areas of low etch-rate. But as the loops begin to expand, they will hit the areas of high etch-rate and the loops will expand more rapidly. The number of nodes in each loop could double, and each loop could in turn generate more loops. The result would be an exponentially growing process that could prove fatal to the continued execution of the program.

Therefore, in a string etching simulation, it is necessary to remove the loops before they become too complex. The string has to be "delooped" every few time-steps. This involves finding the intersection of all the segments in the string and removing the loops from the array of nodes. In *SAMPLE*, this is done approximately every 10 time-steps. Another approach would be to look at the etch-rate encountered and vary the deloop frequency accordingly.

The simulations shown in Figure 6.7 used a fairly small segment size of  $S_{ideal} = 0.04$   $\mu\text{m}$ . It is quite interesting to note that loops are not formed when larger segments are used in the simulation. This is because when the segments are long compared to the etch-distance of each node, the segments will not cross each other to form loops. Instead, there will be a scissoring effect at the converging corners of the profile; this effect can be corrected for without too much difficulty. However, if long segments are used to avoid loops, the overall profile will

---

† The loop formation depends on the advancement distance to the segment length. If the advancement distance is small compared to the segment length, the nodes at the converging corners will be removed before loops can be formed.

no longer be as accurate, since long segments cannot accurately describe a curved surface.

Also note that loops will not be formed in the string algorithm if the surface is advanced by very small time-steps, such that the distance advanced by each node is small compared to the average segment length. However, it is not practical to use very small time-steps, since the computation time will increase dramatically. The computation time is directly proportional to the number of simulation time-steps.

#### **6.2.4. The String Algorithm : Issues In 3D**

The previous discussion has highlighted the difficulty of implementing the string algorithm in 2D. The algorithm is faster and more accurate than the cell methods, but it is difficult to set up and requires careful attention to detail. First, the algorithm requires an ordered series of nodes. Then, the string of nodes has to be modified frequently to keep the string density constant. During this modification, corner effects such as rounding and scissoring must be accounted for; the corners could adversely affect the string profile and the errors so introduced could propagate with the evolution of the string. Finally, loops have to be deleted every few iterations; otherwise, the loops will cause the number of nodes in the string to blow up.

Clearly, implementing the string algorithm in 3D will be quite difficult. Again three main issues must be dealt with. These are : (1) ordering, (2) density, and (3) deloop.

The need to have an ordered mesh complicates considerably the implementation of the string model in three dimensions. In 3D, the boundary between the etched and unetched regions can be approximated by a series of nodes, joined by segments and triangles. The surface is moved by advancing the nodes along vectors, and each direction vector is the average

of the normal vectors to all the triangles that surround a node. In addition, each triangle must consist of an ordered set of nodes; the node ordering implicitly defines the direction of the normal to the triangle. This setup is shown in Figure 6.8, where the normal follows the "right-hand" rule from the sequence of nodes.

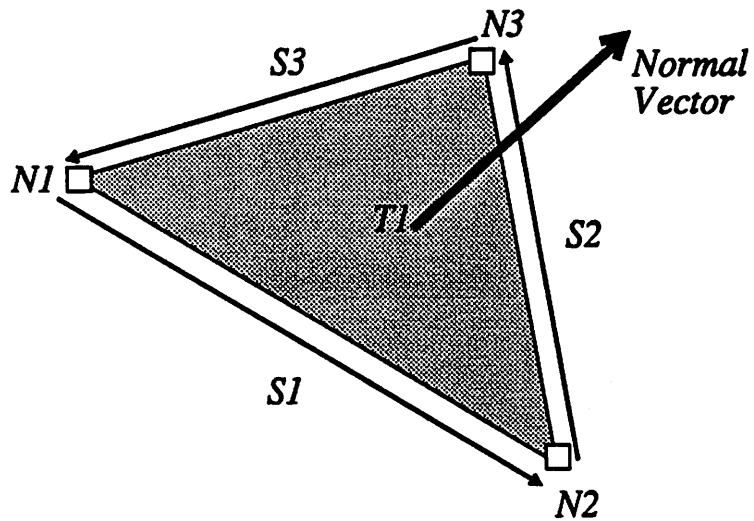
To retain a constant-density mesh, the segments have to be cut or merged if they become too long or too short. This is particularly difficult in an ordered mesh, especially since the orientation or order of the nodes in each triangle have to be rearranged so that the normal vector to the triangle remains unchanged. A great deal of care must be taken to reestablish the connections between the modified segments and the rest of the nodes, segments and triangles of the mesh. †

Additional complications arise from corner effects. In 3D, it is not easy to remesh the surface, yet the 2D experience has shown that without remeshing, i.e., accounting for corners, errors will be introduced. A straight-forward implementation of the string algorithm without correction for corner effects would run, but the results might not be trustworthy. Another complication unique to 3D comes from the approximation of a 3D surface with triangles. Facets are formed whenever triangles are used to approximate an arbitrary curved 3D surface. An example of the 3D faceting problem is shown in Figure 6.9. Here, the true etch surface is cylindrical in shape, and the cylinder is aligned parallel to the x-axis. If four triangles are used to approximate the surface, each of the four triangles will point in different directions. One would hope that the variation in the triangle normal vectors due to the facets of each triangle will cancel out at each node, so that the averaged normal vector remains normal to the true curvature of the surface. But it is not difficult to envision situations in which inaccurate

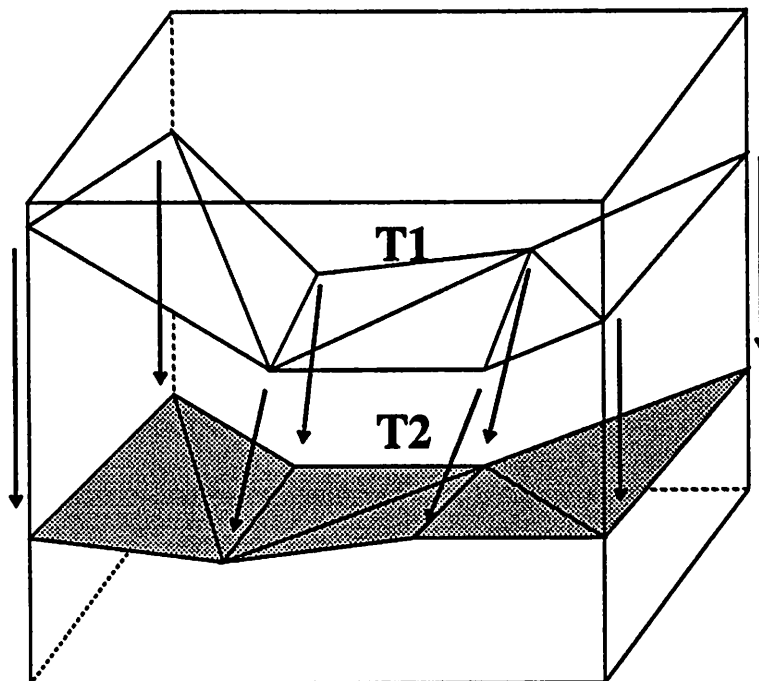
---

† Keeping track of the mesh interconnections is in essence a book-keeping problem. But because of the ordered mesh, the book-keeping is a difficult task and must be error-free.

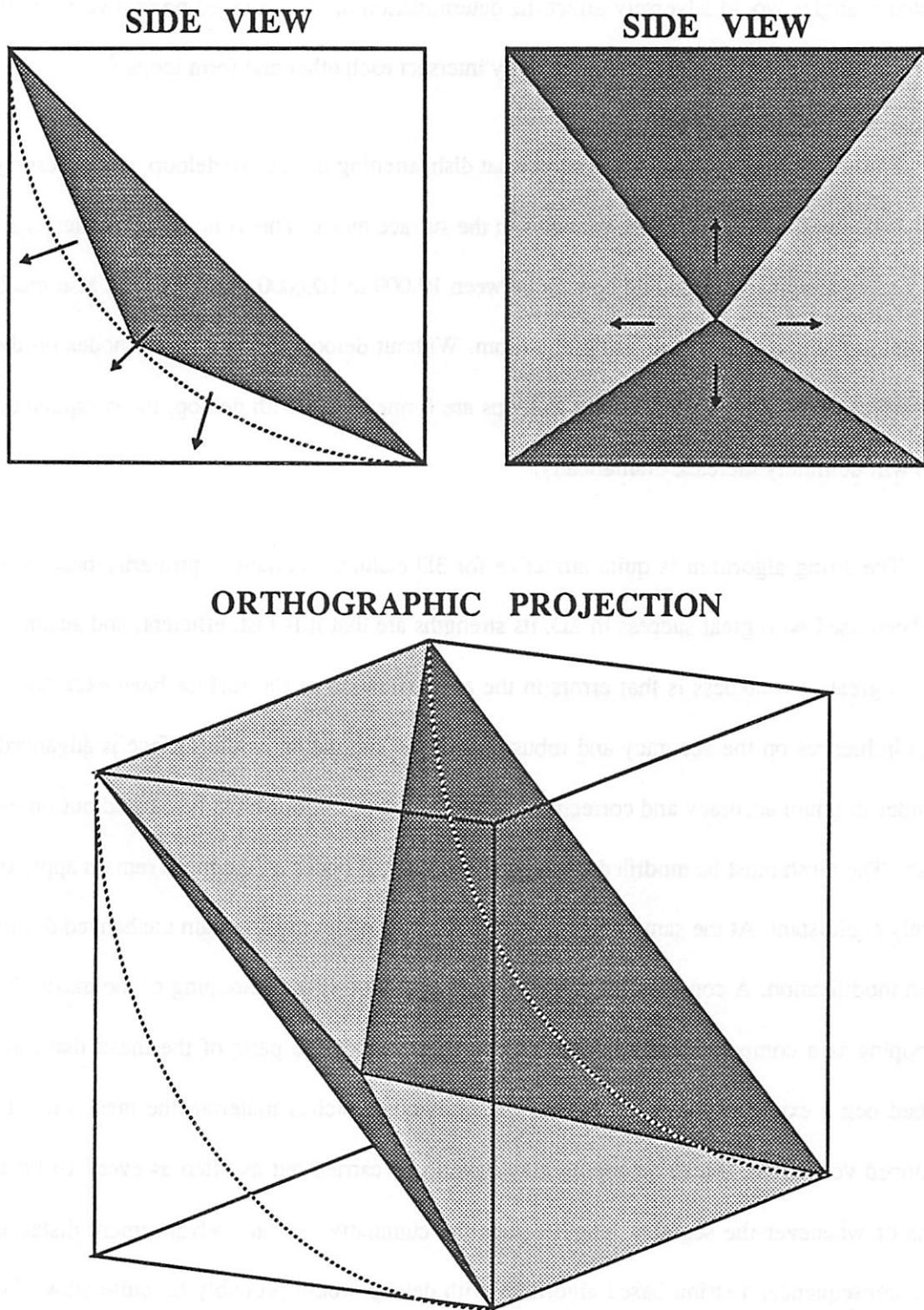
(a)



(b)



**Figure 6.8:** The string model : 3D implementation. (a) The mesh consists of nodes, directed line segments and triangles. The normal vector to the triangle is found from the cross-product of two line vectors. (b) Nodes on the surface are moved along direction vectors; each direction vector is the average of the normal vectors of the triangles surrounding a node.



**Figure 6.9:** 3D Faceting. Facets are formed when triangles are used to approximate a curved surface. The facets could affect the accuracy of the 3D string simulation.

faceted triangles would adversely affect the determination of the averaged normal vectors. In such situations, the triangles could potentially intersect each other and form loops.<sup>2</sup>

The delooping problem is also somewhat disheartening in 3D. To deloop, it is necessary to find the intersections of all the triangles in the surface mesh. This is no trivial matter especially since a typical mesh could contain between 10,000 to 100,000 triangles. But also recall that deloop is essential to the string algorithm. Without deloop, the number of nodes on the 3D surface could grow exponentially as loops are formed. But with deloop, the computation time will definitely increase dramatically.

The string algorithm is quite attractive for 3D etching simulation, primarily because it has been used with great success in 2D. Its strengths are that it is fast, efficient, and accurate. But its greatest weakness is that errors in the approximation of the surface have excessively large influences on the accuracy and robustness of the simulation as the surface is advanced. In order to retain accuracy and correctness, a number of operations must be carried out on the mesh. The mesh must be modified periodically so that the nodes on the mesh remain approximately equidistant. At the same time, the ordering of the nodes must remain unchanged during mesh modification. A considerably more difficult operation is the delooping of the mesh. 3D delooping is a computationally expensive operation. Since the parts of the mesh that have looped begin expanding in all directions into previously etched material, the mesh must be delooped very often. Mesh delooping may have to be carried out as often as every 10 time-steps or whenever the segment length equals the cumulative surface advancement distance. As a consequence, a string-based algorithm with deloop would probably be quite slow. So, unless a good technique can be found to deal with loops in the 3D string algorithm, this algorithm will not be an efficient algorithm for 3D etching.



### 6.3. THE RAY ALGORITHM

The ray algorithm for etching, proposed by Hagouel<sup>3</sup> in 1976, is a method by which nodes are advanced along rays which are refracted at the boundaries of regions of different etch-rates. The algorithm is based upon a rigorous mathematical solution to the least time principle, given by the differential ray equation [5.12]. The differential ray equation is reprinted below.

$$\frac{d}{ds} \left[ \frac{1}{v(x,y,z)} \mathbf{s} \right] = \nabla \left[ \frac{1}{v(x,y,z)} \right] \quad [6.1]$$

To recap, the equation describes the trajectory of a ray through a position-dependent velocity field. This equation could therefore describe the movement of a particle through a velocity field (e.g. air), or the passage of a light ray through an optical media, or, in a case of particular interest, the passage of an etch ray through a material during surface-etching.

There is a close analogy between geometrical optics and etch ray-tracing. In both cases, the medium has a position-dependent velocity field (optical refractive index vs. etch-rate distribution), and it is desired to trace the movement of a front or a surface (optical wave-front vs. etch-boundary) as a function of time. The rays are the trajectories of nodes on the surface, and are orthogonal to the moving surface. Thus, one could use the principles of geometrical optics ray-tracing (e.g. the laws of reflection and refraction) to trace the trajectory of an etch ray, where the etch-material's refractive index is defined as the ratio of the maximum etch-rate  $R_{\max}$  to the etch-rate  $R(x,y,z)$  as a function of position.

$$n_{etch} = \frac{R_{\max}}{R(x,y,z)} \quad [6.2]$$

In fact, the analogy to optics was used by Hagouel<sup>3</sup> to derive the fundamental equation of etch ray-tracing, i.e., Equation [6.1]. It is also worth mentioning at this point that in optics, the eikonal [5.13] and differential ray [5.12] equations are also rigorous solutions to the

electromagnetic wave equation. †

For the purposes of etching simulation, the differential ray equation must be recast in terms of etching quantities. Let  $s$  be the total distance etched by an etch point,  $\mathbf{s}$  the vector along the direction of the etch ray, and  $R(x, y, z)$  the etch-rate in the material as a function of position. Then the equation governing the changes in direction of the etch ray as a function of the etch-rate can be rewritten as

$$\frac{d}{ds} \left[ \frac{1}{R(x, y, z)} \mathbf{s} \right] = \nabla \left[ \frac{1}{R(x, y, z)} \right] \quad [6.3]$$

This equation may be discretized by replacing the differential  $d$  by the variation  $\Delta$ .

$$\frac{1}{\Delta s} \Delta \left[ \frac{1}{R(x, y, z)} \mathbf{s} \right] = \nabla \left[ \frac{1}{R(x, y, z)} \right] \quad [6.4]$$

So, over two points  $P_1$  and  $P_2$ , the relationship between the etch rays is

$$\left[ \frac{\mathbf{s}_2}{R_2} - \frac{\mathbf{s}_1}{R_1} \right] = \nabla \left[ \frac{1}{R} \right] \Delta s \quad [6.5]$$

where  $\mathbf{s}_1$  and  $\mathbf{s}_2$  are the unit ray vectors at the two points. The differential ray equation in this form may be simplified further. After some manipulation (Appendix A.4), the following form is obtained.

$$\mathbf{s}_2 - \mathbf{s}_1 = -0.5 \frac{(R_1 + R_2)}{R_1} \nabla(R) \Delta T + \mathbf{s}_1 \left[ \frac{R_2}{R_1} - 1 \right] \quad [6.6]$$

### 6.3.1. The Algorithm for Advancing the Rays

Equation [6.6] can be used quite nicely to find the trajectory of the etch ray as the ray moves in time. Given a unit direction vector  $\mathbf{s}_1$  at point  $P_1 = (x_1, y_1, z_1)$ , first find the point  $P_2 = (x_2, y_2, z_2)$  by advancing point  $P_1$  along the direction vector  $\mathbf{s}_1$ .

---

† Born & Wolf,<sup>4</sup> Section 3.1

$$\begin{aligned}
 x_2 &= x_1 + (\mathbf{s}_1)_x R(x_1, y_1, z_1) \Delta T \\
 y_2 &= y_1 + (\mathbf{s}_1)_y R(x_1, y_1, z_1) \Delta T \\
 z_2 &= z_1 + (\mathbf{s}_1)_z R(x_1, y_1, z_1) \Delta T
 \end{aligned}
 \tag{6.7}$$

The total distance advanced is just the rate at point  $P_1$  multiplied by the incremental time-step  $\Delta T$ . The next step is to find the new direction vector  $\mathbf{s}_2$  at  $P_2$ . For this, evaluate the etch-rate gradient at  $P_2$  and calculate  $\mathbf{s}_2$  using the discrete ray equation [6.6]. This procedure is shown in Figure 6.10.

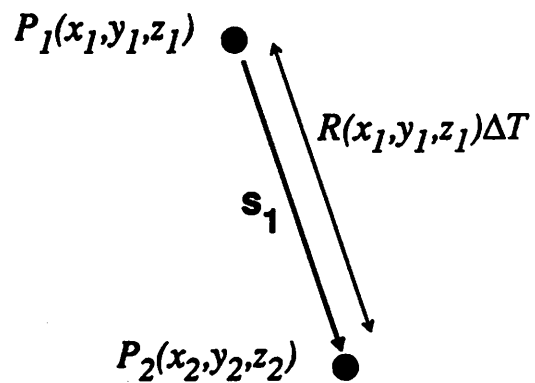
The ray algorithm is depicted in Figure 6.11. The simulation begins by initializing the rays such that they are normal to the initial surface. As shown in Figure 6.11, the surface is typically a smooth horizontal surface, in which case the rays are oriented vertically. The algorithm then proceeds as follows.

- [I] Advance each node along its direction vector.
- [II] Calculate the new direction vector at the new node location using the discretized form of the differential ray equation, Equation [6.6].
- [III] Proceed to [I] and repeat until the total etch time has been reached. Then construct the final etch front out of the end-points of the rays.

### 6.3.2. Implementation of the Ray Algorithm

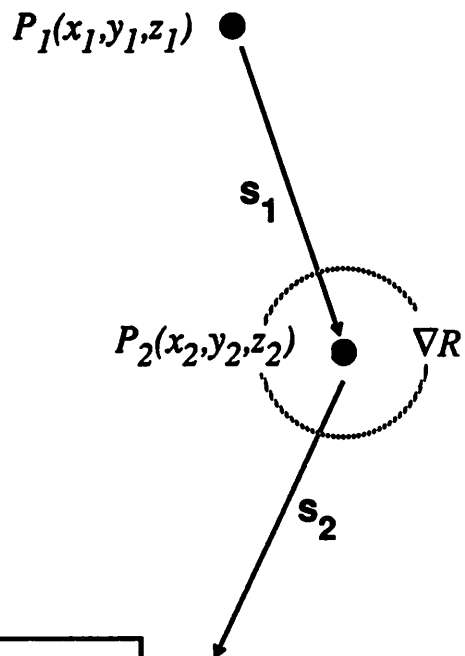
The ray algorithm is unique in that each ray is independent of all other rays. The trajectory of a ray is dependent only on the ray's previous trajectory and the local etch-rate, so it is not necessary to keep track of the connections between different rays. Little memory is needed for individual ray calculations. (In fact, Hagouel's initial 2D ray simulations were done interactively on a desk calculator!)<sup>1</sup> In direct contrast, the string algorithm picks up trajectory information from the local surface; this information can only be obtained from a

### 1. ADVANCE THE POINT



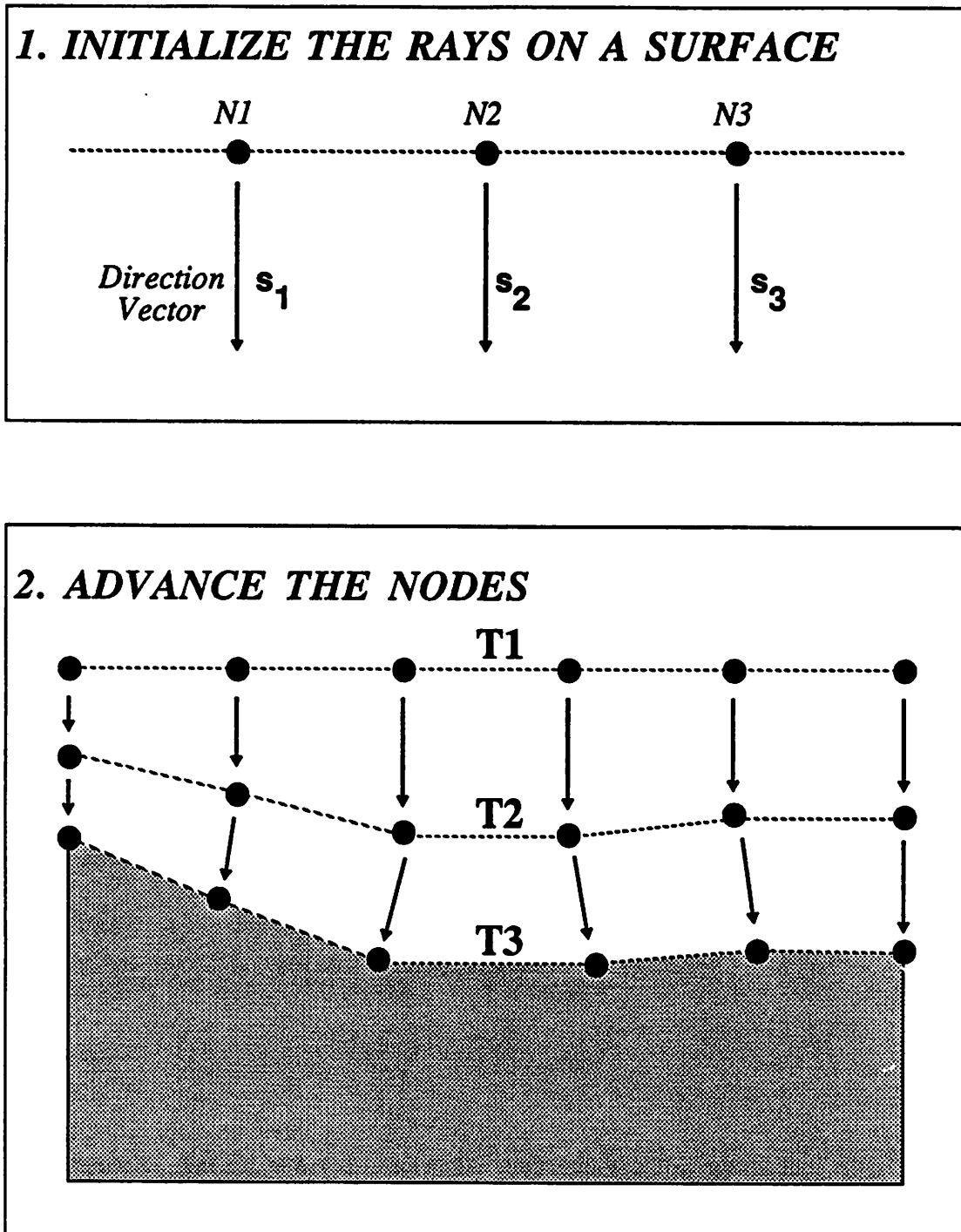
$$\Delta S = R(x_1, y_1, z_1) \Delta T$$

### 2. CALCULATE NEW VECTOR



$$\mathbf{s}_2 = -0.5(1 + R_2/R_1)\nabla R\Delta T + \mathbf{s}_1(R_2/R_1)$$

**Figure 6.10 :** The Ray Algorithm : At every time-step, first advance the point, then calculate the new direction vector at the new point using the discretized ray equation, Equation [6.6].



**Figure 6.11 :** The Ray Algorithm. Begin by initializing the rays normal to the initial surface. Then during each step, advance the nodes and find the direction changes due to etch-rate refraction. Finally, reconstruct the surface from the end-points of the rays.

connected surface mesh which in turn requires access to the coordinate data of each node. Because of the independence of the rays, the rays can be implemented as a linked list of nodes, where each node retains information on the current position and direction vector of the node. The linked list arrangement is optional; it allows for greater flexibility in the number of rays in the simulation volume.

### 6.3.3. Testing the Ray Algorithm

#### 6.3.3.1. Etching with An Exponential Etch-Rate Function

Figure 6.12 shows an etching simulation using a simple exponential function

$$R(x,y,z) = e^{-4x^2} \quad [6.8]$$

This function resembles the development etch-rate from electron-beam exposure.

Using this function in the discretized ray equation [6.4], the differential equation describing the change in the vector of the ray becomes

$$\begin{aligned} \frac{1}{\Delta s} \Delta \left[ \frac{1}{R(x,y,z)} \mathbf{s} \right] &= \nabla \left[ \frac{1}{R(x,y,z)} \right] \\ &= \nabla \left[ \frac{1}{e^{-4x^2}} \right] \\ &= 8x e^{4x^2} \end{aligned} \quad [6.9]$$

where  $\mathbf{l}$  is the unit vector in the  $x$ -direction. As before,  $\Delta s = R(x,y,z)\Delta T$ . According to Equation [6.9], the gradient of the inverse of the etch-rate is a vector pointing in the positive  $x$ -direction for positive  $x$ . In this case, a ray starting out from a point with a positive  $x$  coordinate will curve away from the  $x = 0$  plane; its direction vector will have an increasingly larger  $x$ -component as the ray marches forward. Similarly, because the gradient is negative for negative  $x$ , a ray in the negative  $x$ -quadrant will have a vector with a negative  $x$

## EXPONENTIAL RAY ETCH

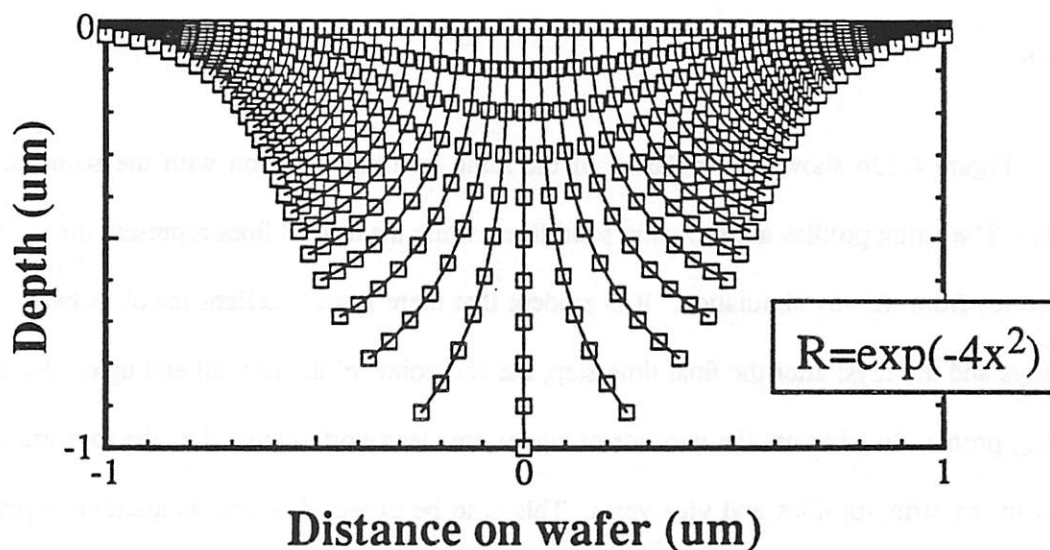


Figure 6.12a : The Ray Algorithm : 51 etch rays in an exponential etch-rate function. The simulation uses 10 time-steps of 0.1 seconds each. The etch front is formed from the end-points of the rays at every time-step.

## 51-pt STRING ETCH

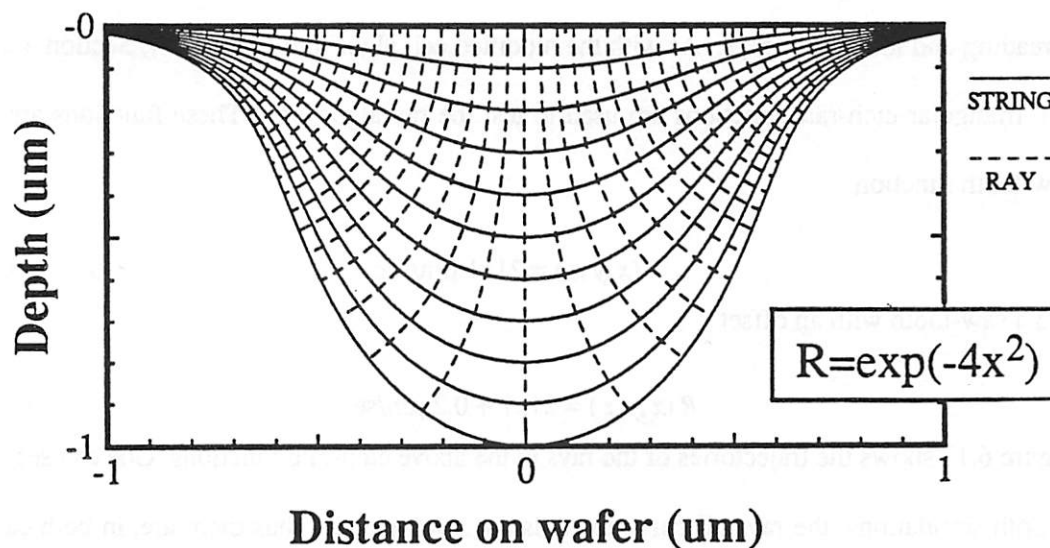


Figure 6.12b : String-simulated etch profile in an exponential etch-rate function. The simulation uses 10 time-steps of 0.1 seconds each. The rays from (a) have been overlaid. Note that the end-points of the rays do coincide with the string profiles.

component, and the ray again curve away from the  $x = 0$  plane. This behavior is indeed seen in the ray-tracing simulation of Figure 6.12a. As expected, the rays bend away from the  $x = 0$  plane.

Figure 6.12b shows a simulation of the same etch-rate function with the string algorithm. The string profiles are shown in solid lines, while the dashed lines represent the ray trajectories from the ray simulation. It is evident that there is an excellent match between the strings and the rays; after the final time-step, the end-points of the rays all end up on the final string profile. In addition, the two sets of curves are clearly orthogonal, i.e., the rays are normal to the string-profiles and vice-versa. This is to be expected, since, as mentioned previously, the rays are the normal trajectories to the etch fronts.

### 6.3.3.2. Ray-Spreading and Looping In Triangular Etch-Rates

The use of a triangular etch-rate function also provides interesting information about the spreading and looping of rays. As with the modified cell algorithm (Chapter 4, Section 4.5.2), two triangular etch-rate functions are used to test the ray algorithm. These functions are the saw-tooth function

$$R(x,y,z) = 2|x| \mu\text{m/sec} \quad [6.10]$$

and a saw-tooth with an offset

$$R(x,y,z) = 2|x| + 0.2 \mu\text{m/sec} \quad [6.11]$$

Figure 6.13 shows the trajectories of the rays in the above etch-rate functions. One can see that in both simulations, the rays all curve towards the  $x = 0$  plane. Thus there are, in both cases, ray-scarce regions, i.e., regions in which no or few rays pass. This is an inherent weakness of the ray algorithm; there are certain regions which may not be reached by the first choice of initial locations on the surface.<sup>1</sup>



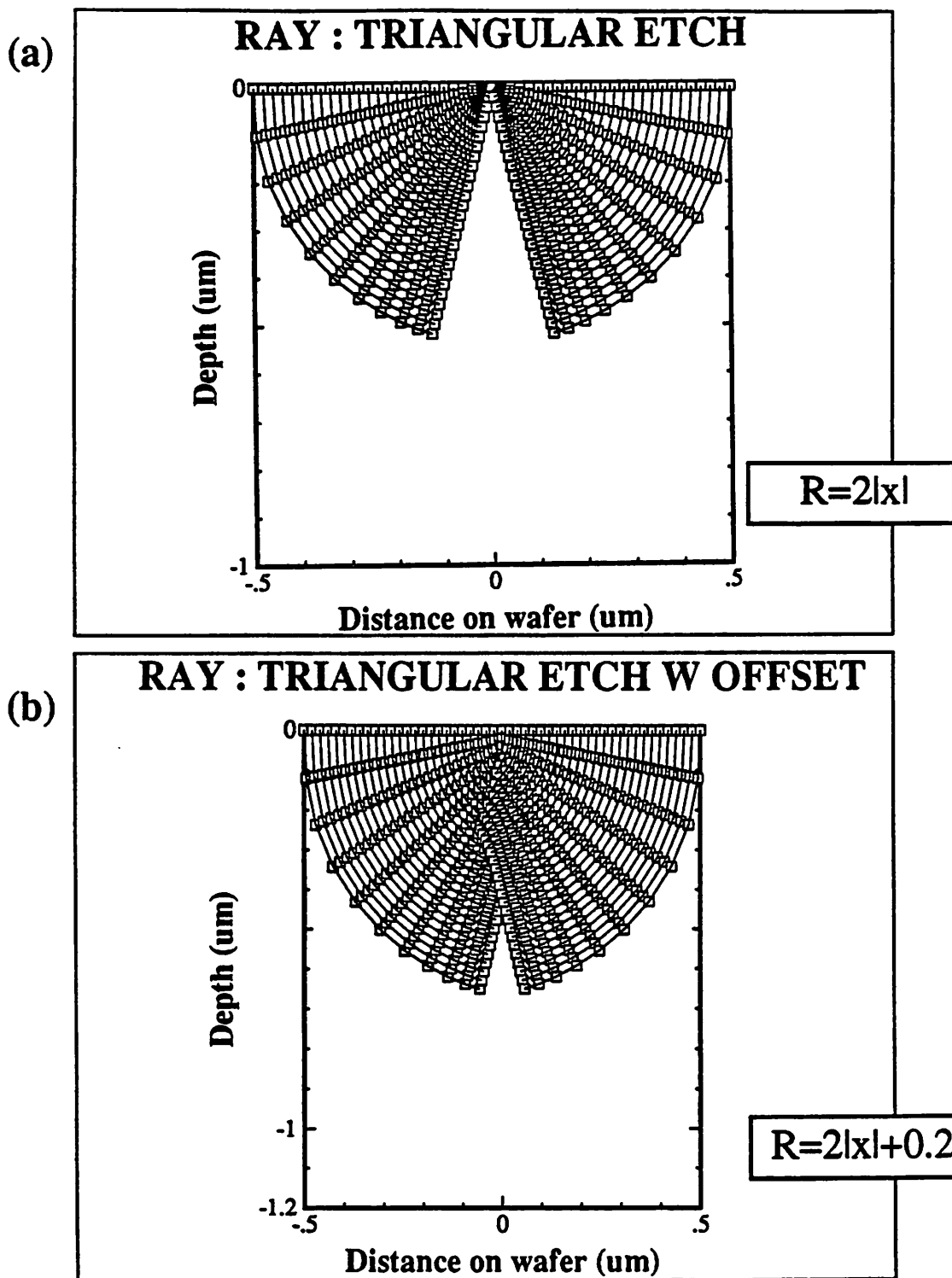


Figure 6.13 : 2D ray etching using triangular etch-rate functions, (a)  $R=2|x|$  and (b)  $R=2|x|+0.2$ . Simulation proceeds for 10 0.1-second time-steps. All the 51 rays have curved inward, leaving a ray-scarce region for large  $|x|$  values. In addition, in (b) some rays have crossed.

In addition, in the simulation with the offset rate function (Figure 6.13b), some of the rays have even crossed each other in the center. Physically, these rays are now in the etched region and lag behind the actual etch front. Thus there will be loops in the surface formed by the end-points of the rays. The formation of loops was also seen in the string algorithm. But the ray-formed loops do *not* turn inside-out and expand outwards in all directions as in the string algorithm. It is also worth noting that loops are not formed in a volume etching simulation. The modified-cell simulations of the etch-fronts for the triangular etch-rate functions above were shown in Figures 4.4b and 4.5b respectively.

### 6.3.3.3. Ray-Spreading and Looping in Sinusoidal Etch-Rates

The etch-rate function

$$R(x,y,z) = e^{-4x^2}(1.05 - \cos(2\pi z)) \quad [6.12]$$

is a function that resembles the standing-wave etch-rate distribution in photoresist development. This function has  $x$  and  $z$  dependence and thus makes for a good test of the ray algorithm.

Figure 6.14 shows the ray trajectories for a 3-second etching simulation with 30 0.1-second time-steps and 300 0.01-second time-steps respectively. There are a total of 51 rays in each simulation; the rays are initially placed uniformly between  $x = -0.4$  and  $x = 0.4$ . The rays are seen to move down and outwards into the material. Most of the rays have bent back towards the initial surface; some of these have also crossed each other, forming loops. There are also ray-scarce regions in the center of the simulated area.

It is worth noting that the ray trajectories in both the 30 and 300 time-step simulations trace out similar etch fronts, that is, the surface traced by the envelope of the rays are similar

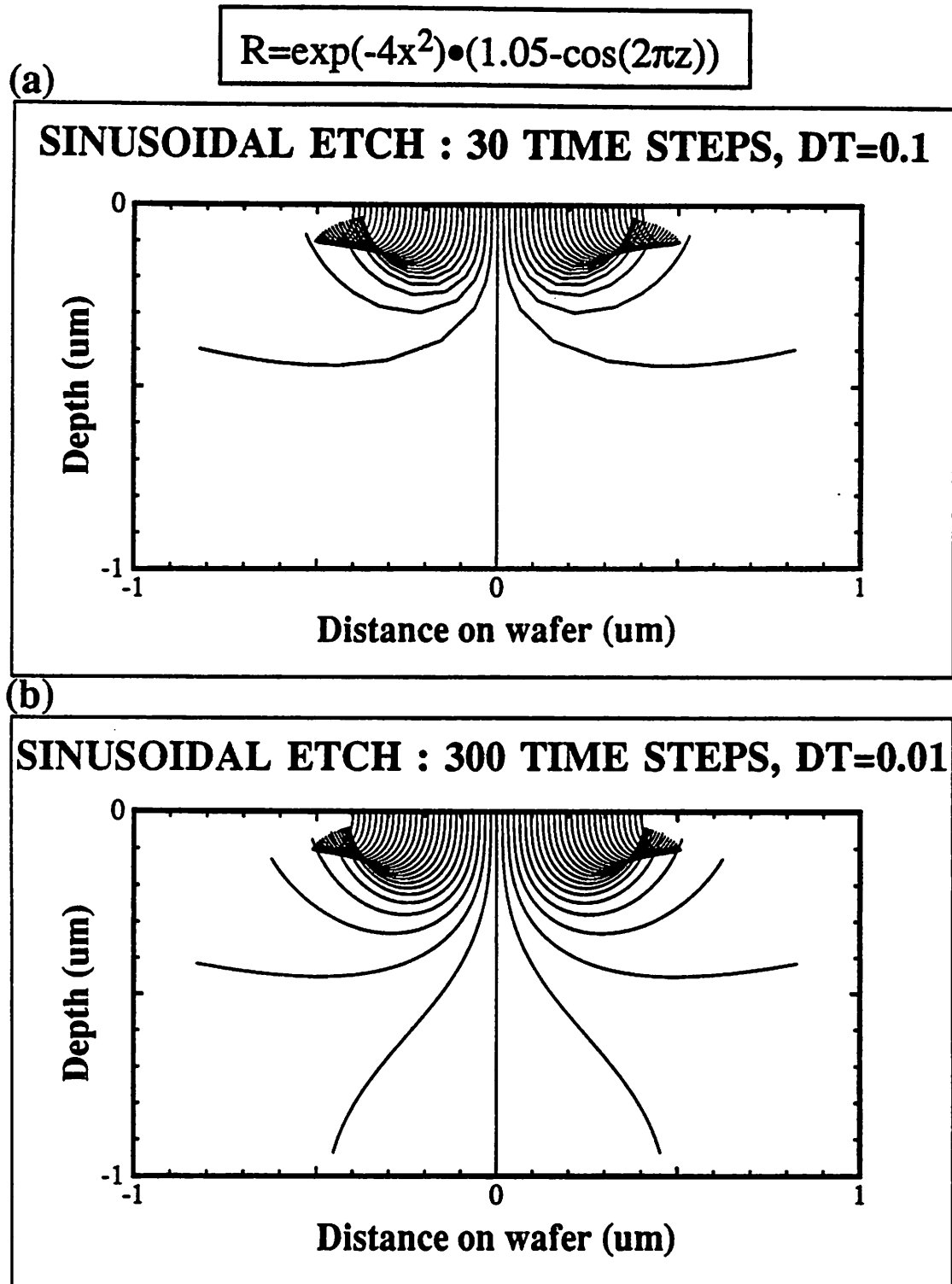


Figure 6.14 : 2D ray etching using a sinusoidal etch-rate function. Simulation proceeds for a total of 3 seconds, with (a) 30 time-steps of 0.1 seconds and (b) 300 time-steps of 0.01 seconds each. The difference in the size of the time-step causes the ray trajectories to change.

in Figures 6.14a and 6.14b. However, the actual paths of the rays differ in the two figures. This is due to the fact that the accuracy of any discretized simulation will increase if more points are sampled, or in this case, if more iterations are used. In the 30 time-step simulation, the differential ray equation is only evaluated 30 times. As a result, the simulation is not as accurate, and the rays do not curve as much as the 300-time-step rays.

Another interesting aspect of the ray algorithm is seen in Figure 6.15, where the simulations have been run with a higher-frequency sinusoidal etch-rate function.

$$R(x, y, z) = e^{-4x^2}(1.05 - \cos(8\pi z)) \quad [6.13]$$

In the 30-time-step simulation, shown in Figure 6.15a, almost all the rays are grouped at the initial surface. Even the ray at the center has not moved very far from the surface. In contrast, the rays in the 300-time-step simulation shown in Figure 6.15b have fascinating smoothly curved trajectories.

Notice that in Figure 6.15b, the rays are curled up around depths of  $z = 0$ ,  $z = -0.25$  and  $z = -0.5 \mu\text{m}$ . It is no coincidence that the etch-rate is slowest at these depths. As can easily be seen from Equation [6.13] above, the etch-rate function is periodic in the  $z$ -coordinate, with a period of  $0.25 \mu\text{m}$ . Thus, at the surface  $z = 0$ , and at depths of integer multiples of the  $0.25 \mu\text{m}$  period (e.g.  $z = -0.25 \mu\text{m}$ ,  $z = -0.50 \mu\text{m}$ ), the etch-rate is at a minimum value. This is also seen clearly in Figure 6.16, which shows the contour map of this particular etch-rate function.

Comparing the etch ray trajectories of Figure 6.15b with the contour map of the etch-rate in Figure 6.16, one can see that some of the etch rays have been trapped inside regions of relatively low etch-rate. This behavior of the rays can be understood with the aid of the discretized ray equation [6.5], rewritten below.

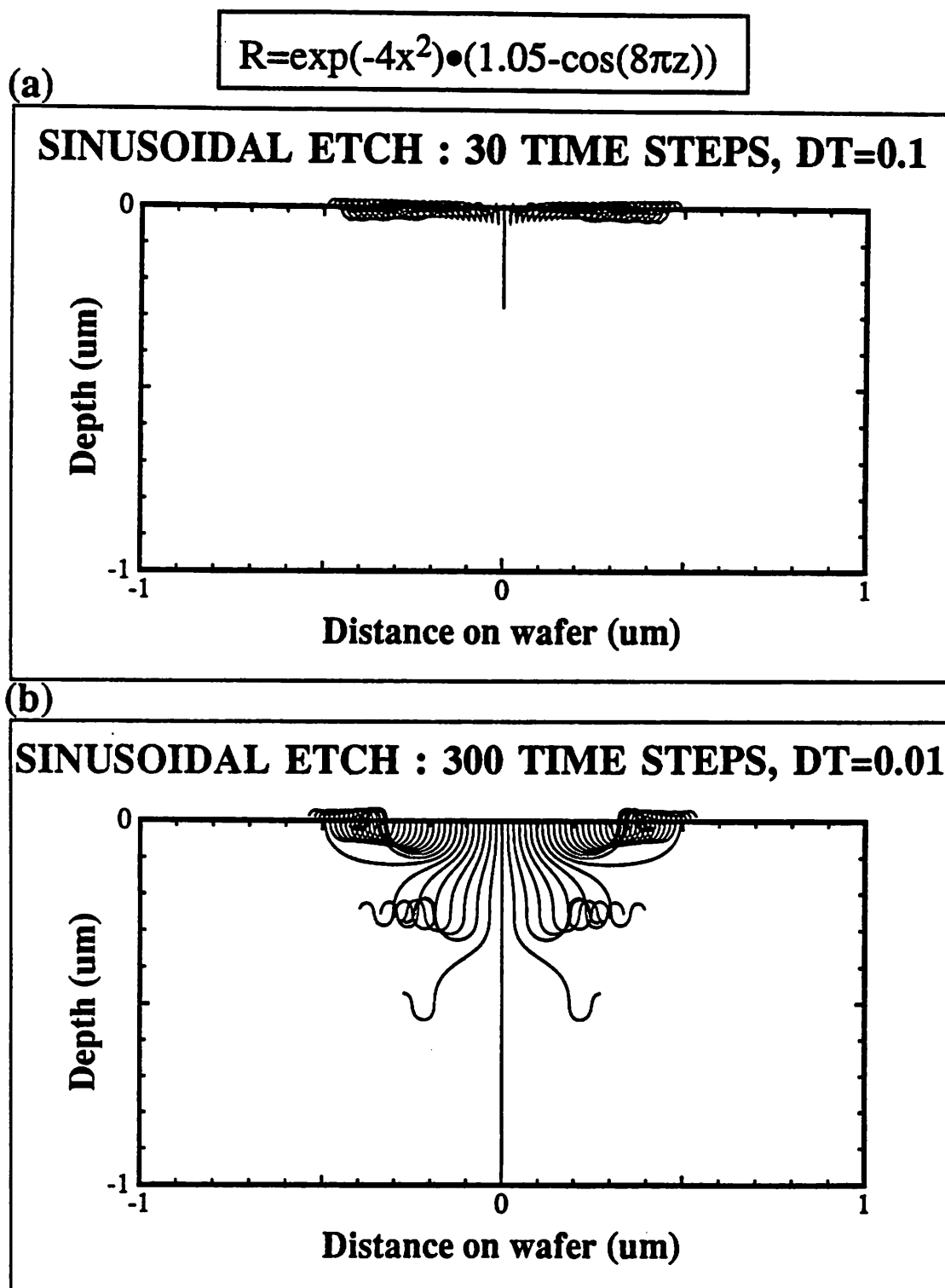


Figure 6.15 : 2D ray etching using a high-frequency sinusoidal etch-rate function. Simulation proceeds for a total of 3 seconds, with (a) 30 time-steps of 0.1 seconds and (b) 300 time-steps of 0.01 seconds each. The rays in (a) are the result of incorrect reflections.

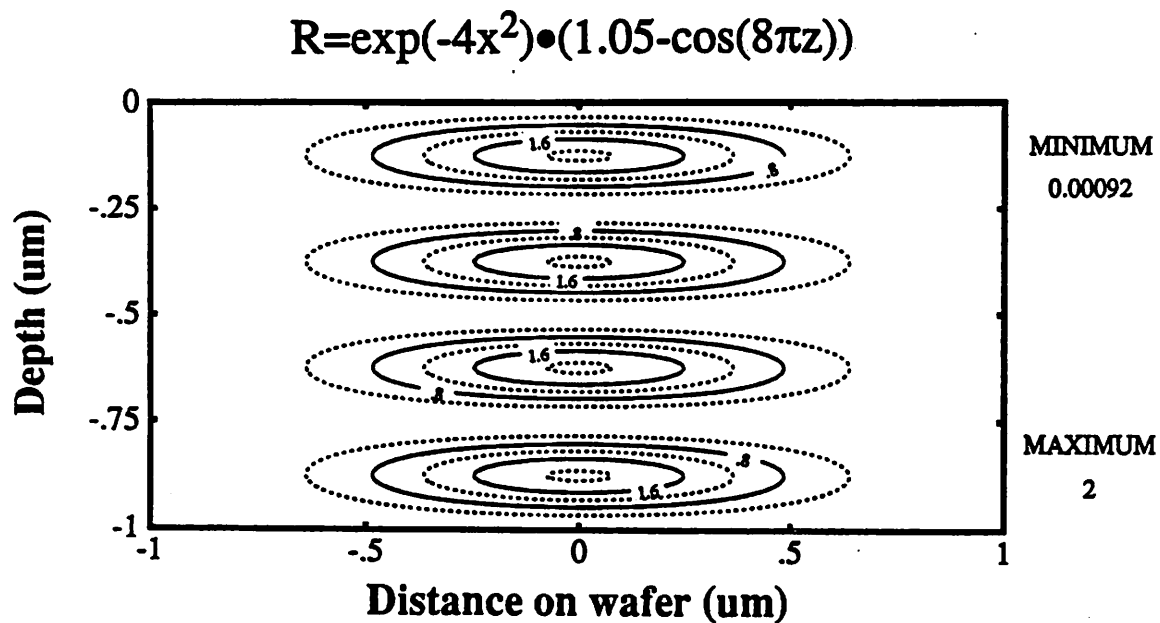


Figure 6.16: 2D etch-rate contours for a high-frequency sinusoidal etch-rate function. The etch-rate is at a minimum at  $z=0.0, -0.25, -0.5, -0.75$  and  $-1.0$  um.

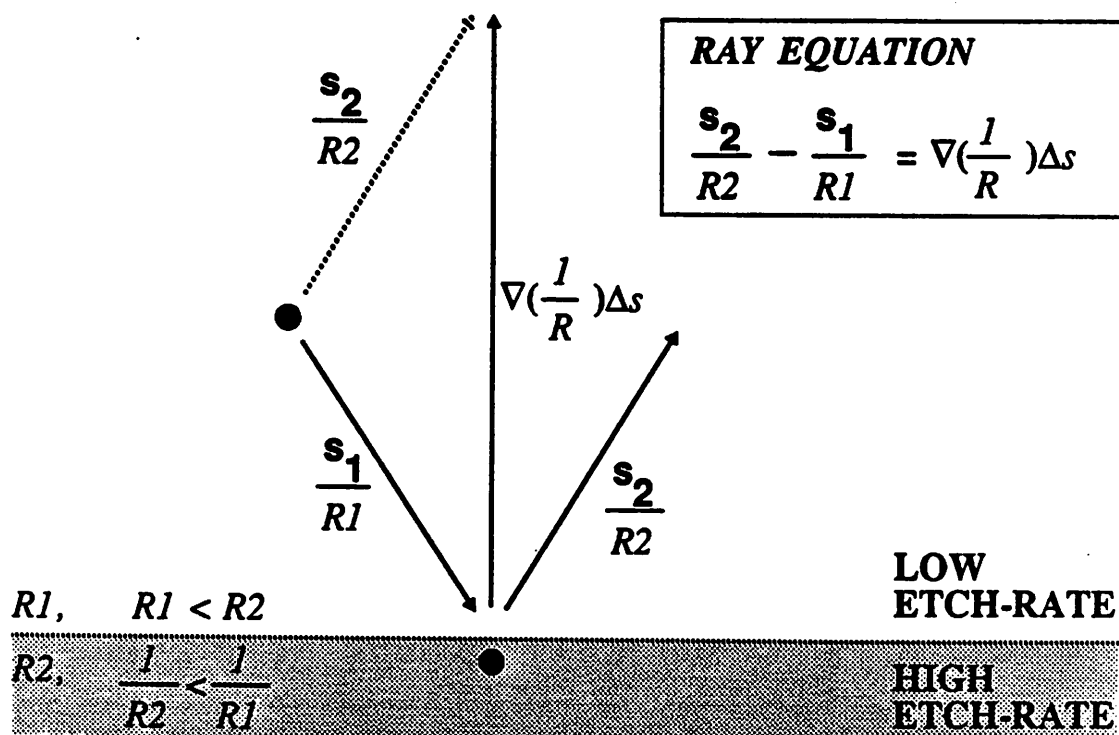


Figure 6.17: Reflection of an etch-ray. This occurs for certain angles of incidence if the etch-rate  $R_1$  in region 1 is less than the etch-rate  $R_2$  in region 2.

$$\frac{\mathbf{s}_2}{R_2} = \frac{\mathbf{s}_1}{R_1} + \nabla \left[ \frac{1}{R} \right] \Delta s \quad [6.14]$$

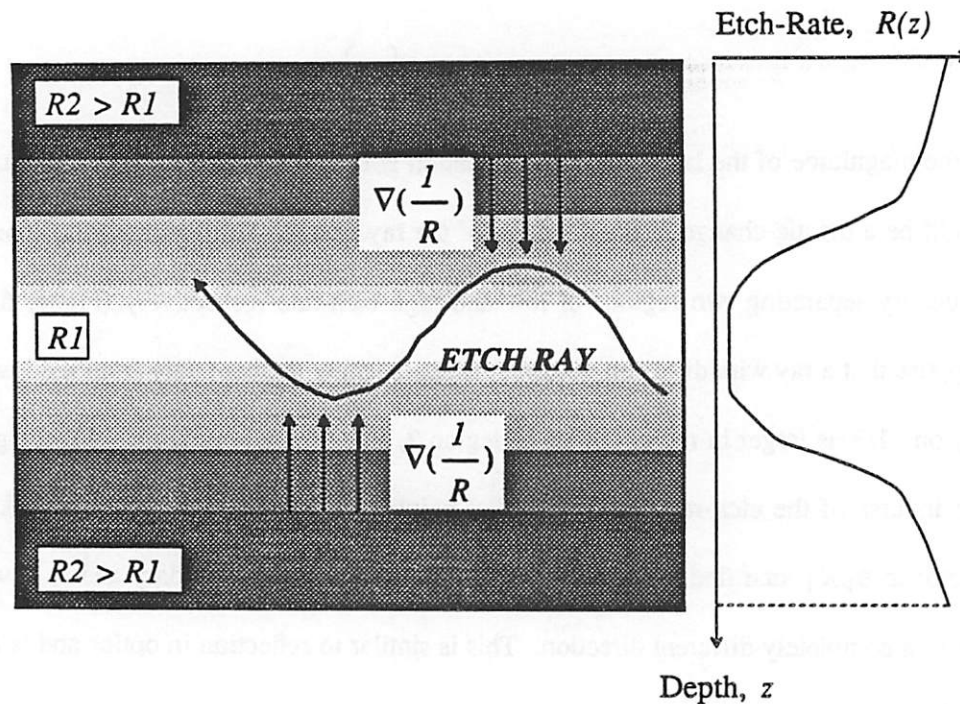
If the magnitude of the last term in this equation is larger than that of the vector  $\mathbf{s}_1/R_1$ , there could be a drastic change in the direction of the ray vector. To illustrate this, consider a flat boundary separating two regions of low and high etch-rate respectively (Figure 6.17). Now suppose that a ray with direction vector  $\mathbf{s}_1$  is approaching the boundary from the low etch-rate region.  $1/R$  is larger in region 1 than in region 2, so as shown in Figure 6.17, the gradient of the inverse of the etch-rate is a vector that points upwards from the boundary. Adding this vector to  $\mathbf{s}_1/R_1$ , one finds that the ray  $\mathbf{s}_2$  has bounced off the boundary and is now shooting off in a completely different direction. This is similar to reflection in optics and is the analog of Snell's Law.

The situation just described is really an over-simplification. One constraint on the discretized ray equation [6.4] is that the variation  $\Delta$  must remain small so that the differential equation [6.3] is obeyed. As a consequence, the change in the vectors  $\Delta(\mathbf{s}/R)$  must remain small, and abrupt changes in direction such as described in Figure 6.17 are not allowed.†

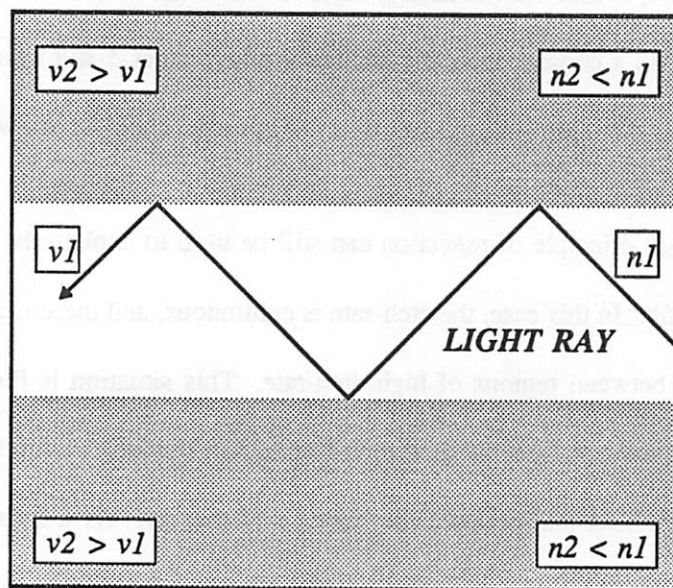
Nevertheless, the basic principle of reflection can still be used to explain the curling of the etch rays in Figure 6.15b. In this case, the etch-rate is continuous, and there is a region of low etch-rate sandwiched between regions of high etch-rate. This situation is illustrated in Figure 6.18. A ray that starts out from the low etch-rate region towards the high etch-rate region will encounter a gradient that opposes the passage of the ray. As a result, the ray curves back towards the low etch-rate region. This gradual ray reflection occurs on both sides

---

† If reflection does occur at a boundary, the laws of geometrical optics say that the angle of incidence is equal to the angle of reflection. This law is clearly not obeyed in the example of Figure 6.17. This is because the differential ray equation only applies for continuous etch-rate  $R(x,y,z)$ . At abrupt boundaries, the ray equation has to be disregarded in favor of Snell's Law of refraction.



**Figure 6.18 :** Etch-Ray Trapping : A ray is trapped in a low etch-rate region sandwiched between high-etch-rate regions. The gradual change in etch-rate causes the ray to bend smoothly.



**Figure 6.19 :** An optical waveguide. A material with high refractive index (low wave velocity) is sandwiched in between materials of low refractive index (high wave velocity). Rays entering at certain incident angles will be trapped and guided by the waveguide.



of the low-etch-rate valley, so, as a result, the ray is trapped in the region with low etch-rate. However, the ray can still move within that region. Thus, the ray traces out an S-curved trajectory similar to those observed in the simulation of Figure 6.15b. This situation is, in fact, analogous to the waveguide structure in geometrical optics, illustrated in Figure 6.19. The optical waveguide is constructed out of a region of high refractive index (low wave velocity) sandwiched by layers of low refractive index (high wave velocity). The light rays are reflected at the boundaries, and as a result, the rays are trapped and can thus propagate along the waveguide.

The strange behavior of the rays in Figure 6.15a can also be traced back to the reflection of rays at etch-rate boundaries. Note that in the simulation shown in Figure 6.15a, most of the rays are curved smoothly. But some rays in the center have sharp jagged trajectories. These rays have undergone sharp reflections in the low-etch-rate region about the surface. The abrupt reflection described earlier in Figure 6.17 has occurred. This abrupt reflection of rays is undoubtedly due to the large time-step of 0.1 seconds. Since the time-step is large, the etch-distance  $\Delta s$  is also correspondingly large, and so, the magnitude of the gradient vector term in Equation [6.14] is large, leading to an abrupt change in the ray vector  $\mathbf{S}$ . But as just discussed, the abrupt reflection, with large  $\Delta(\mathbf{S}/R)$ , is a violation of the ray equation! Therefore, the large time-step combined with blind application of the differential ray equation has led to incorrect ray trajectories.

#### 6.3.4. The Ray Algorithm : Issues in 3D

The 3D ray algorithm is not very different from the 2D ray algorithm. The ray algorithm is very easy to implement in 3D; all that is required is the addition of an extra dimension to the calculations. Figure 6.20 shows a 3D simulation of ray-etching with the analytic

sinusoidal function below.

$$R(x, y, z) = e^{-4(x^2+y^2)}(1.05 - \cos(8\pi z)) \quad [6.15]$$

The trajectories of 121 rays were traced through 300 time-steps of 0.1 seconds each, resulting in the jelly-fish-like profiles shown.

Figure 6.20 is an excellent example that illustrates the advantages and disadvantages of the ray algorithm. The ray algorithm is easy to implement in both 2D and 3D. It is accurate and fast, although some care must be taken to limit the size of the time-step in order to avoid abrupt loop reflection. The major advantage of the ray algorithm is that the rays are independent of each other; the rays do not depend on the local etch-front for calculation of their trajectories. Thus, unlike the string algorithm, the rays are not affected by incorrect surfaces or facets. The weakness of the ray algorithm, however, is that certain regions may not be reached by the first choice of initial ray locations on the surface. It is also quite difficult to reconstruct an etch-front from the end-points of rays. This is especially true when loops are formed during the simulation.

#### 6.4. SUMMARY : 3D SURFACE-MOVEMENT ALGORITHMS

Both the string and ray methods are surface movement algorithms that track the surface of the material as it is being etched. The three-dimensional string method approximates the etching boundary between the etched and unetched regions with a series of nodes, joined by segments and triangles. The surface is moved by advancing the nodes along vectors calculated by averaging the normal vectors to all the triangles that surround a node. The ray algorithm, on the other hand, does not keep track of the 3D surface - it merely traces selected points on the time-evolving etch surface. In this algorithm, the nodes are advanced along rays which are refracted at the boundaries of regions of different etch-rates.

$$R = \exp(-4(x^2 + y^2)) \cdot (1.05 - \cos(8\pi z))$$

300 TIME STEPS

DT = 0.01 seconds

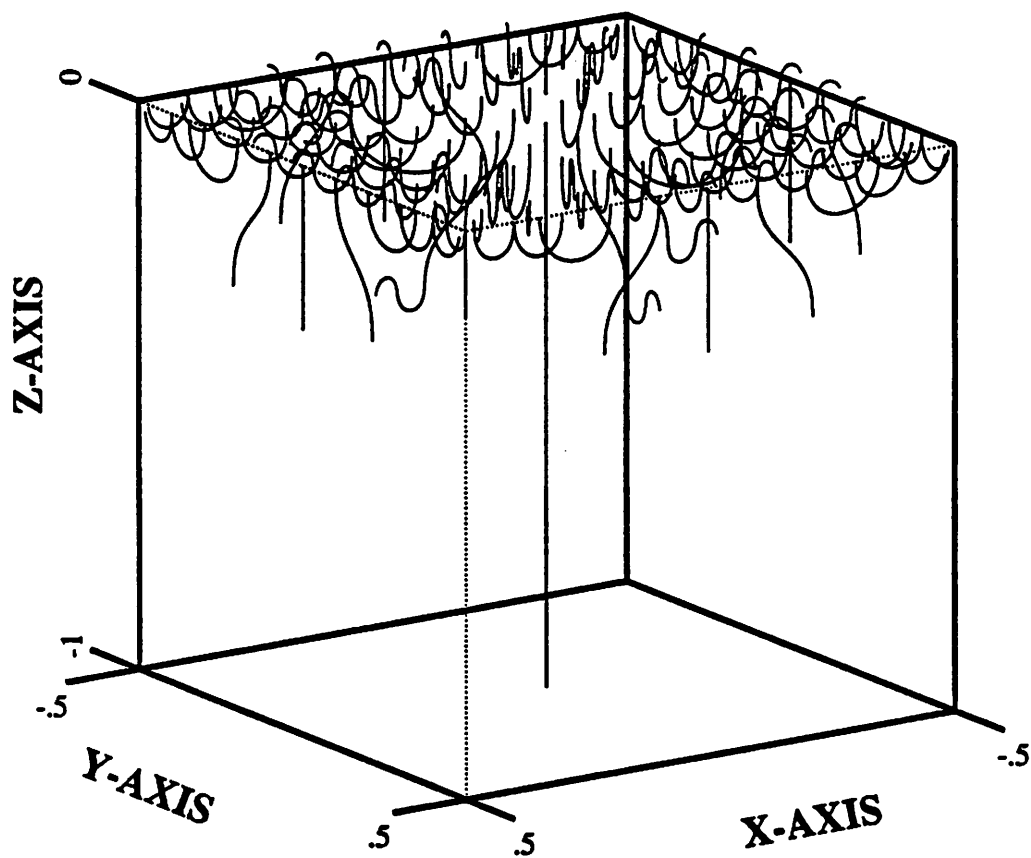


Figure 6.20: 3D ray etching using a high-frequency sinusoidal etch-rate function. 121 etch rays are traced for 3 seconds with 300 time-steps of 0.01 seconds each.

The string method is difficult to set up in 3D, due to the logistical problem of keeping track of all the nodes, segments and triangles that make up the etching boundary. Additional difficulties are encountered because of the necessity of adding and deleting new nodes so that the triangular mesh is neither too dense nor too sparse. The ray method is easy to implement in 3D, but its weakness is that there are certain regions that are not reached by the choice of the initial rays. Thus, it is not easy to reconstruct the etch-surface formed by the endpoints of the rays.

The string and the ray algorithms are, in a sense, complementary algorithms, in that they are both based on solutions to the least time principle. The string mesh represents the eikonal wavefront surfaces, while the rays are the normal vectors to the wavefronts. In the ray algorithm, the rays or advancement vectors are calculated using the differential ray equation. In contrast, the string calculates the advancement vectors from the local surface. This method of calculating the advancement vectors is a key difference between the ray and string methods. Comprehensive testing of the two approaches has shown that the vector calculation method used in the string algorithm is more error-prone than that used in the ray algorithm. In the string algorithm, the advancement vectors calculated from the local surface will be inaccurate if the local surface is incorrect or inaccurate. In contrast, in the ray algorithm, the vectors are calculated from the local etch-rate distribution and are independent of the local surface. The ray vectors are also independent of each other. Therefore, because of the difference in the way the solution to the least-time principle is implemented, the ray algorithm is less sensitive to error and thus more accurate than the string algorithm.

However, the ray algorithm is not a complete solution to the original problem of tracing the time-evolving etch front. The ray algorithm accurately calculates the trajectory of rays, but there are often insufficient rays to adequately describe the entire surface. Somehow,

additional rays must be added to ray-scarce regions. One way of doing this is to connect the rays with a string-like mesh. This method combines the advantages of both the string and ray methods; the string-like mesh covers the entire etch surface, while the rays accurately describe the normals to the etch surface. This combined ray-string approach is described in the following chapter.

## REFERENCES

1. R.E. Jewett, P.I. Hagouel, A.R. Neureuther, T. Van Duzer, "Line-Profile Resist Development Simulation Techniques," *Polymer Eng. Sci.*, vol. 17, no. 6, pp. 381-384, June 1977.
2. E.W. Scheckler, June 1989. Personal communication, work in progress.
3. P.I. Hagouel, "X-ray Lithographic Fabrication of Blazed Diffraction Gratings," *Ph.D. Dissertation*, University of California, Berkeley, 1976.
4. M. Born, E. Wolf, in *Principles of Optics, Sixth Edition*, Pergamon Press, London 1980.

## **CHAPTER 7**

### **THE RAY-STRING ALGORITHM : 2D IMPLEMENTATION ISSUES**

#### **7.1. INTRODUCTION**

The ray-string approach is a powerful algorithm for the simulation of etching that combines the advantages of both the ray and the string algorithms. As with the ray and the string algorithms, the ray-string algorithm is also based on a rigorous mathematical solution to the principle of least time. In this approach, the etch surface is approximated by a string-like mesh of nodes and segments. Like the ray algorithm, the surface is moved by advancing the nodes along rays determined by the local etch-rate in the material. The string-like mesh is used to keep track of the connection between the nodes; as the nodes move further apart or closer together, nodes can be added or deleted.

This combination offers a number of advantages. As a surface-advancement algorithm, the ray-string approach is fast and accurate, and has modest memory requirements. The ray-technique of calculating the direction vectors of the nodes is independent of the local etch surface. This avoids the introduction and propagation of errors from incorrect portions of the mesh. The ray-string algorithm thus is more accurate and less error-prone than the string algorithm. At the same time, the interconnectivity conferred by the string-mesh avoids the formation of ray-scarce regions, and allows for easy reconstruction of the etch-surface.

The concept of the ray-string algorithm itself is not new. Moniwa et.al.<sup>1</sup> recently introduced a 3D photoresist simulator based on this approach. However, Moniwa did not address the issues of accuracy and efficiency, which are of considerable importance in a surface-

advancement algorithm. It is these very topics that shall be the emphasis of this chapter.

As was seen in the previous chapter, both the string and ray algorithms needed a number of adjustments in order to produce accurate and correct simulation results without using excessive amounts of computer memory and time. The same is true of the ray-string algorithm. It too must deal with the tradeoff between accuracy and speed. There are really two general issues that have to be addressed. First, is it possible to improve the calculation of the ray trajectories so that an optimum balance between accuracy and speed is obtained? And second, how do the mesh operations affect the accuracy and execution speed of the simulation? These issues will be addressed and new improvements to the ray-string algorithm will be discussed in the following pages.

## **7.2. IMPLEMENTATION OF THE RAY-STRING ALGORITHM IN 2D**

In order to investigate the accuracy vs speed tradeoffs in the ray-string algorithm, a 2D version of the algorithm has been implemented in the C programming language. The data structure consists of two separate linked lists of nodes and segments. As in the ray algorithm, each node retains information on its current position and direction vector. But now, each node also contains pointers to its adjoining segments. Each segment too is linked by pointers to two nodes. Thus, the connection between the nodes and the segments is two-way.

## **7.3. THE ALGORITHM FOR SURFACE ADVANCEMENT**

The procedure for advancing the surface mesh in the ray-string algorithm is very similar to that of the string and the ray algorithms. The simulation begins by creating and initializing a surface consisting of a number of interconnected nodes. Each node is initialized with a ray



or direction vector normal to the initial surface. The algorithm then proceeds as follows (Figure 7.1):

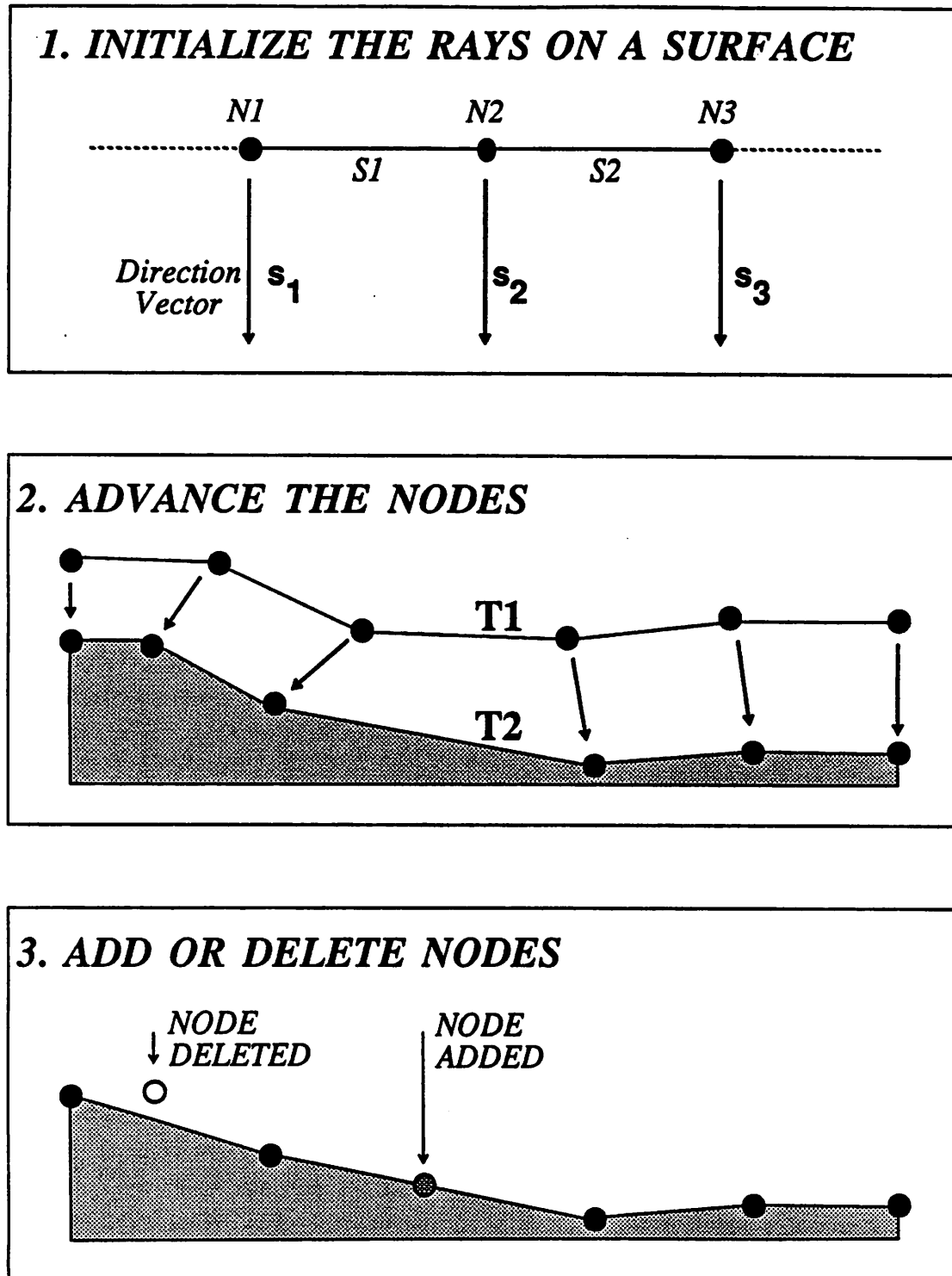
- [I] Advance each node on the mesh along its direction vector. The distance advanced is equal to the local etch-rate multiplied by the incremental time-step.
- [II] Calculate the new direction vector at the new node location using the discretized form of the differential ray equation, Equation [6.6].
- [III] Add nodes in regions where the mesh has expanded, and delete nodes in regions of contraction.
- [IV] Proceed to [I] and repeat until the total etch-time has been reached.

#### **7.4. SELECTION OF THE SIZE OF THE TIME-STEP**

There is, however, a problem with the algorithm outlined above. The ray-algorithm simulations described in Chapter 6 have shown that the ray trajectories are sensitive to the size of the time-step taken to advance the rays. If the time-step is too large, abrupt ray reflections could take place (Figure 6.15a, Section 6.3.3.3). The trajectory of the rays would then be incorrect. To avoid such gross errors in the ray trajectories, one could, of course, use very small time-steps. But this improvement in accuracy would come at a cost of increased computation time. So, how does one choose the time-step so that the rays can be calculated both accurately and efficiently?

##### **7.4.1. Avoiding Abrupt Ray Reflection**

One rule of thumb in selecting the time-step is to just use a time-step  $\Delta T$  that has been determined by trial and error. For example, in the ray simulation with the high-frequency



**Figure 7.1 :** The Ray-String Algorithm. Begin by initializing the rays normal to the initial surface. Then during each step, advance the nodes and find the direction changes due to etch-rate refraction. After each surface advancement, add or delete nodes where necessary.

sinusoidal etch-rate shown in Figure 6.15, a time-step of  $\Delta T = 0.01$  seconds appears to be a good choice. But if the vertical frequency of the etch-rate is increased, the  $\Delta T$  chosen above might not be sufficient to ensure accuracy. In other words, the trial-and-error selection of the time-step must be repeated for all the different etch configurations to be simulated.

An even better approach for selecting the time-step can be derived from the ray equation. Begin with the discretized ray equation [6.5].

$$\frac{\mathbf{s}_2}{R_2} - \frac{\mathbf{s}_1}{R_1} = \nabla \left[ \frac{1}{R} \right] \Delta s \quad [7.1]$$

To simplify the equation above, assume that the etch-rates are of the same order of magnitude within a small sphere of radius  $\Delta s$ . Then,  $R_1 \approx R_2 \approx R$ , and Equation [7.1] becomes

$$\frac{\mathbf{s}_2}{R} - \frac{\mathbf{s}_1}{R} = \nabla \left[ \frac{1}{R} \right] \Delta s = \frac{-1}{R^2} \nabla(R) R \Delta T \quad [7.2]$$

Simplifying further, one obtains

$$\mathbf{s}_2 - \mathbf{s}_1 = -\nabla(R) \Delta T \quad [7.3]$$

The relevance of this equation to the selection of the time-step can be understood with the aid of Figure 7.2. It is not difficult to see that if the magnitude of the last vector term in [7.1] is kept small, then the change in the direction of the vectors will be small too. But when this last term is large, then the rays could change direction abruptly. To avoid such abrupt reflections in the ray trajectories, the magnitude of the last term in [7.3] should be limited. If the gradient term is allowed to have a magnitude only less than 1.0, then the change in direction of the vectors will be at most  $60^\circ$ . Thus, the conditions for avoiding reflection can be stated as follows.

$$|\nabla(R)| \Delta T < 1 \quad [7.4]$$

This equation is a concise statement of the conditions under which rays will *not* undergo abrupt reflection. Abrupt ray reflection will not occur if (i) the time-step  $\Delta T$  is small, and (ii) the magnitude of the gradient of the etch-rate is small (i.e., if the variation of the etch-rate

**RAY EQUATION  
(SIMPLIFIED)**

$$s_2 - s_1 = -\Delta(R)\Delta T$$

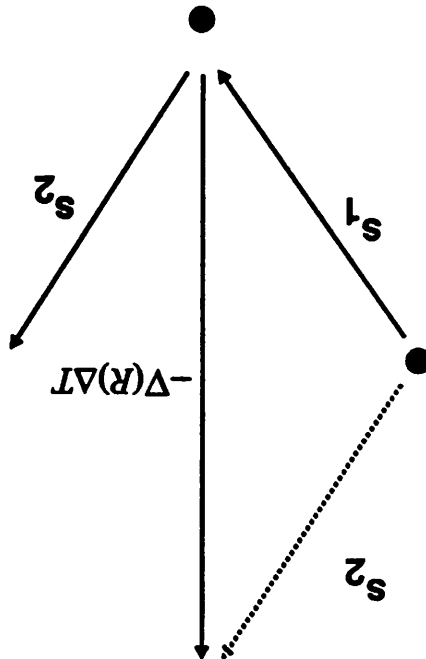


Figure 7.2 : Vector Checking : The vector difference  $\Delta(R)\Delta T$  should be kept small to avoid abrupt ray reflection.

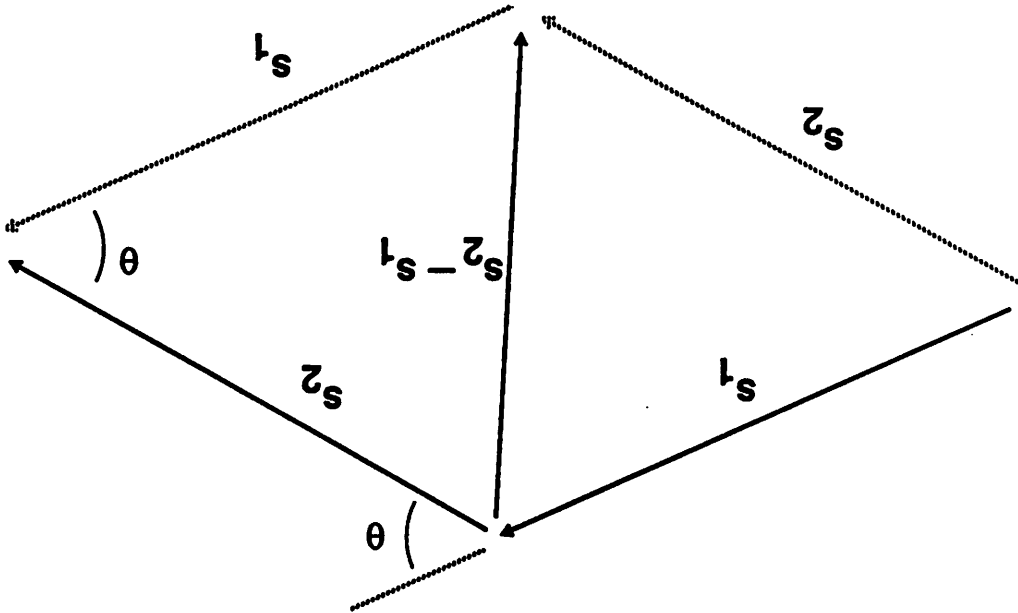


Figure 7.3 : Vector Checking : The angle  $\theta$  has to be kept small so that there is little change in the direction of the rays.

with distance is small).

Equation [7.4] states that the optimum size of the time-step should scale inversely with the maximum slope of the etch-rate distribution. For example, the maximum slope of the sinusoidal etch-rate function [6.13] is  $\partial R / \partial z = 8\pi$ , so a good time-step size for that particular function would be  $\Delta T = (\partial R / \partial z)^{-1} = 0.04$  seconds. In typical photoresist development simulations, though, the etch-rate distribution consists of discrete values. In such distributions, it is not practical to search for the maximum etch-rate gradient. However, if the etch-rate distribution varies periodically in the vertical direction due to standing waves, then it is possible to construct a criteria similar to that of [7.4]. If  $Z_T$  is the period of the standing waves, and  $R_{\max}$  is the maximum etch-rate in the volume of the photoresist, then the condition

$$\frac{R_{\max}}{Z_T / (2\pi)} \Delta T < 1 \quad [7.5]$$

is sufficient to ensure that the rays do not bounce abruptly.

Equation [7.5] can be tested on the etch-rate distribution shown in Figure 2.3. In this case, the vertical period is  $Z_T \approx 0.125 \mu\text{m}$ , while the maximum etch-rate is  $0.11 \mu\text{m}/\text{sec}$ . The, according to Equation [7.5], the maximum time-step size that can be tolerated is  $\Delta T = 0.18$  seconds. This number actually turns out to be a very good choice. As shall be seen later in the chapter,  $\Delta T = 0.1$  seconds produces good simulated profiles, whereas with  $\Delta T = 0.5$  seconds, the rays are seen to bounce back to the initial surface.

#### 7.4.2. Recursive Vector Checking

Equation [7.4] is not itself a rigorous condition for accuracy, because of it is based on the assumption that the etch-rates are approximately equal at the two points traversed by the

ray in a single time-step. However, it can serve as a good rule of thumb for selecting the size of the time-step.

The accuracy of the ray-calculation can be further refined by subdividing the initial choice of time-step. In the "recursive vector checking" procedure, the size of the time-step is selected by limiting the change in the direction of the ray vectors to a few degrees.

Recall that the implicit constraint on the discretized ray equation [6.4] is that the vector variation  $\Delta(\mathbf{s}/R)$  must be kept small, so that the variance  $\Delta$  is approximately equal to the differential  $d$ . Now, keeping the change in the vectors  $\Delta(\mathbf{s}/R)$  small is almost equivalent to ensuring that the change in unit vectors  $\Delta\mathbf{s}$  is kept small. So, the condition for satisfying the differential ray equation becomes

$$|\mathbf{s}_2 - \mathbf{s}_1| \ll 1 \quad [7.6]$$

The relationship between the unit vectors  $\mathbf{s}_1$ ,  $\mathbf{s}_2$  and the difference between the two vectors,  $\mathbf{s}_2 - \mathbf{s}_1$ , is shown in Figure 7.3. If the magnitude of  $\mathbf{s}_2 - \mathbf{s}_1$  is kept small, then clearly the angle between the vectors  $\mathbf{s}_1$  and  $\mathbf{s}_2$  will be small as well. It is possible to show that

$$|\mathbf{s}_2 - \mathbf{s}_1| = 2 - 2\cos^2\theta \quad [7.7]$$

So, if the magnitude of the vector difference is limited to 0.1 ( $|\mathbf{s}_2 - \mathbf{s}_1| \leq 0.1$ ), the angle  $\theta$  between the vectors will be at most  $4^\circ$  ( $\theta \leq 4^\circ$ ).

The procedure for recursive vector checking may now be outlined. First, define a minimum recursive length  $|\Delta\mathbf{s}|_{\min} = |\mathbf{s}_2 - \mathbf{s}_1|_{\min} = l_{\min}$  (e.g.  $l_{\min} = 0.1$ ) and an initial step-size  $\Delta T$ .

- [I] Evaluate the length of the vector  $|\Delta\mathbf{s}| = |\mathbf{s}_2 - \mathbf{s}_1|$ .
- [II] If the length of the vector  $|\Delta\mathbf{s}|$  is greater than  $l_{\min}$ , divide the time-step in two. Recalculate the new vector  $\mathbf{s}_2$ . Return to [I] and continue until

$$|\Delta \mathbf{s}| = |\mathbf{s}_2 - \mathbf{s}_1| < l_{\min}$$

- [III] If the time-step is now such that the condition  $|\Delta \mathbf{s}| < l_{\min}$  is satisfied, advance the ray to the new node according to the size of the current time-step. Return to [I], and repeat recursively.

The pseudo-code for this recursive procedure is as follows.

---

```

advance_both(point, vector, dt)
{
    /* advance the point along its vector */
    advance_the_point(point, newpoint, vector, dt)

    /* calculate the new vector at the new point */
    advance_the_vector(point, newpoint, vector, newvector)

    /* if vector difference is too large, dt is too large */
    vector_difference = magnitude ( newvector - vector )
    if (vector_difference > 0.1) then
        advance_both(point, vector, 0.5*dt)
        advance_both(point, vector, 0.5*dt)
    else
        point = newpoint
        vector = newvector
    end if
}

```

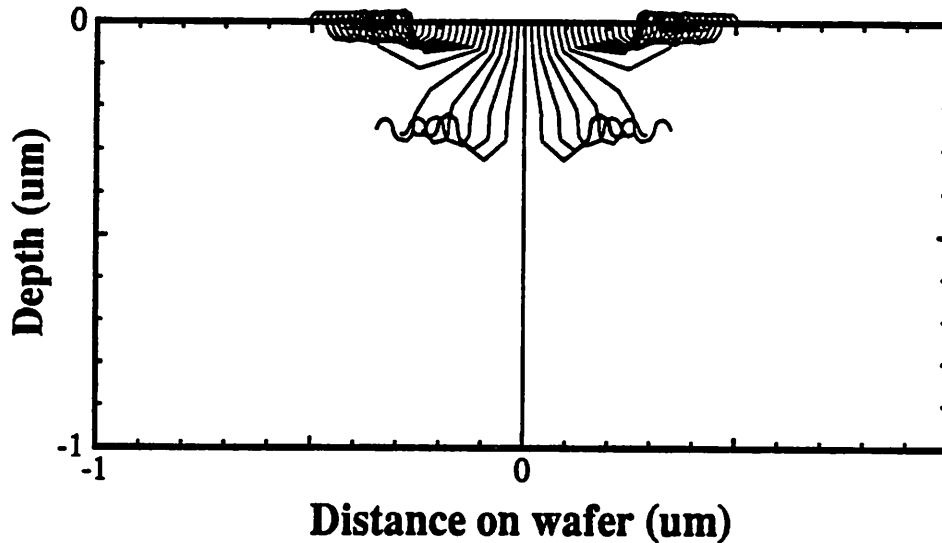
---

Examples of the effect of recursive vector checking on the ray simulations are shown in Figure 7.4. The simulation conditions are similar to those of Figure 6.15, except that in this set of simulations, the recursive vector procedure is used to limit the direction changes of the rays.

In Figure 7.4a, a recursive vector length of 0.1 is used, and as in Figure 6.15a, the ray simulation is over 30 time-steps of 0.1 seconds each. The results are indeed quite satisfactory; the ray trajectories do compare favorably with the 300-time-step simulation of Figure 6.15b.

$$R = \exp(-4x^2) \cdot (1.05 - \cos(8\pi z)), \text{ Recursive Length} = 0.1$$

(a)

**SINUSOIDAL ETCH : 30 TIME STEPS, DT=0.1**


(b)

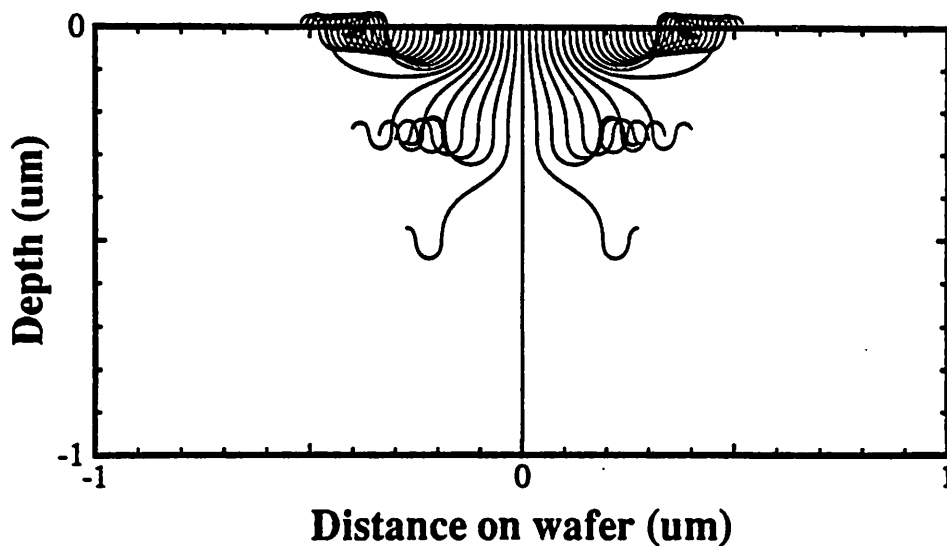
**SINUSOIDAL ETCH : 300 TIME STEPS, DT=0.01**


Figure 7.4 : 2D ray etching using a high-frequency sinusoidal etch-rate function. The use of recursive vector checking with recursive length of 0.1 increases the accuracy of the simulations and avoids incorrect reflections of the rays.



The rays do have sharp-edged trajectories, but this is due to the large time-step and the fact that the intermediate points calculated during the recursive procedure are not saved. The ray trajectories are not quite as accurate as those of Figure 6.15b, but at least they are much more accurate than those in the original 30-step simulation in Figure 6.15a.

Figure 7.4b shows the result of the ray simulation with 300 time-steps of 0.01 seconds each. As before, recursive vector checking is used with a recursive length of 0.1. This time, the ray trajectories appear to be similar to those of Figure 6.15b. The recursive checking procedure has had little effect on the accuracy of the simulation.

The computation times for the simulations of Figure 6.15 and 7.4 are tabulated in Table 7.1. The simulation of Figure 6.15a runs the fastest, but in this particular simulation, the rays trajectories are quite incorrect. When recursive-vector checking is used with a large time-step, the computation time almost doubles, but the ray trajectories become relatively accurate. Clearly, for large time-steps, recursive vector checking is well worth the price paid in computation time. But for small time-steps, recursive vector checking has little or no effect on the accuracy or on the computation time. So, in effect, recursive vector checking is a procedure that increases the accuracy of the simulations only when it is necessary to do so.

Figure	Time-Step Size	Vector Recursion	Computation Time
6.15a	0.1 sec	No	5 sec
6.15b	0.1 sec	No	44 sec
7.4a	0.01 sec	Yes	11 sec
7.4b	0.01 sec	Yes	46 sec

Table 7.1 : The effect of recursive vector checking on computation time.

### 7.4.3. NEW MODIFICATIONS TO THE SURFACE-ADVANCEMENT METHOD

The constraints on the vectors implicit in the ray equation has led to two new procedures that are useful for optimizing the accuracy and the efficiency of the simulations. The first is a rule-of-thumb which allows for the selection of a "good" time-step in which abrupt ray-reflection is avoided. This procedure, however, is of use only when the etch-rate distribution is periodic. There is no such constraint on the second procedure, which is a recursive method for accurately tracking the trajectory of the rays in some given time interval. This recursive procedure is of particular use in situations where the etch-rate is varying rapidly in only small localized regions. In this case, it is not efficient to choose a global time-step optimized for the fast-varying etch-rate regions. It is far better to use a larger time-step suited for the more widespread low-varying etch regions; the recursive vector procedure becomes useful primarily in the fast-varying etch-rate regions.

These two procedures can be easily added to the algorithm for surface-advancement. The first step is, as before, to select an initial time-step. If the etch-rate is periodic, then Equation [7.4] or [7.5] could be used to select an optimum time-step. Otherwise, the time-step could be chosen based on the average or the maximum etch-rate in the volume of the material being etched. Once the initial time-step has been chosen, the surface can be advanced as described in Section 7.3. The recursive procedure would cover steps [I] and [II] in Section 7.3. The algorithm for the surface advancement then becomes :

- [I] Advance each node and vector recursively according to the procedure outlined in Section 7.4.2.
- [II] Add nodes in regions where the mesh has expanded, and delete nodes in regions of contraction.

[III] Proceed to [I] and repeat until the total etch-time has been reached.

## 7.5. MESH OPERATIONS

Another issue of considerable importance in the ray-string algorithm concerns step [II] in the algorithm outlined above. As in the string algorithm, a number of operations have to be repeatedly carried out on the ray-string surface mesh, so that the mesh remains both accurate and correct. These mesh operations, and their effect on the accuracy and efficiency of the ray-string simulations will be described in some detail in this section.

### 7.5.1. Mesh Modification

Surface-advancement algorithms in general suffer from the disadvantage that as the surface mesh moves in time, the nodes on the mesh will move farther apart or closer together. The goal of mesh-modification is to modify the mesh so that all the nodes on the mesh are equidistant.† As a result, the simulation will become more accurate and also more efficient.

In the ray-string algorithm, the connection between the nodes on the mesh is maintained by a list of segments. Therefore, in order to retain a constant node density in the mesh, segments have to be cut up if they become too long. Or, if the segments become too short, these segments have to be deleted.

The mesh should ideally be modified after every surface advancement. To do this, a range of acceptable segment lengths should first be defined. As in the string algorithm, one

---

† One alternative method for modifying the mesh is to tie the density of nodes to the etch-rate, i.e., to use more nodes where the mesh is moving fast, and to use fewer nodes in slow-moving regions. This method, while useful in some applications, does not appear suited for the simulation of photoresist development, where the etch-rate changes rapidly with distance.

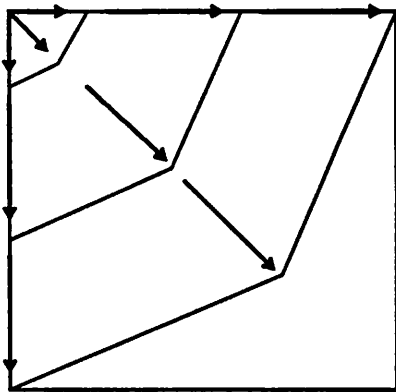
can define a nominal or ideal segment length  $S_{ideal}$  and maximum ( $S_{max}$ ) and minimum ( $S_{min}$ ) acceptable lengths. Then, each segment in the list is examined in turn. If the segment length  $S$  is greater than the maximum allowable length, i.e.,  $S > S_{max}$ , the segment must be cut up. Or if the segment length is less than the minimum allowable value, that is,  $S < S_{min}$ , the segment is deleted or merged. After each cut or merge, the interconnections between segments and nodes in the mesh must be updated.

### 7.5.1.1. Segment Cutting

During the etch simulation, nodes on a mesh could move far enough apart that certain areas on the mesh will become sparsely populated. The mesh is then no longer accurate in the sparse region. As an example, Figure 7.5 depicts the growth of a 2D etch front during a uniform circular etch. Without mesh modification, the nodes on the circular front will move farther apart, and as a result, the mesh becomes faceted. From Figure 7.5a, it is quite clear that the simulation results will be inaccurate in the sparse areas of the mesh. In contrast, if new nodes are added whenever the segments grow too large, then a more ideal circular profile will be formed, as shown in Figure 7.5b.

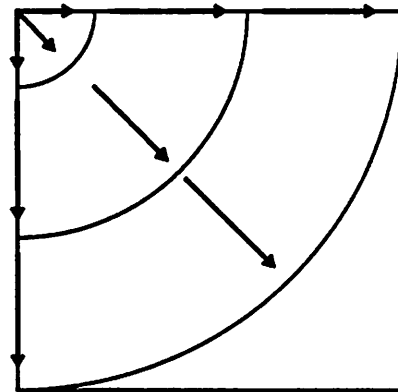
To add new nodes to the mesh, the segment lengths should be checked after every advancement of the surface. If a segment is too long, then it should be cut in half. This can most easily be done using linear interpolation, where, as shown in Figure 7.6a, the new node is placed on the original segment, equidistant from the two original nodes. The direction vector of the new node is similarly an average of the direction vectors of the two original nodes. An alternate method, described briefly in Section 6.2.3, is to add nodes using a polynomial fit. This method, which is used in *SAMPLE*, does result in smoother and more accurate results. However, it is important to realize polynomial fitting cannot be easily applied to 3D

**WITHOUT  
MESH MODIFICATION**



(a)

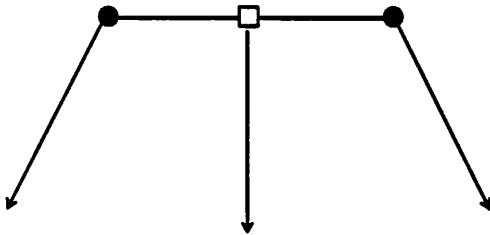
**WITH  
MESH MODIFICATION**



(b)

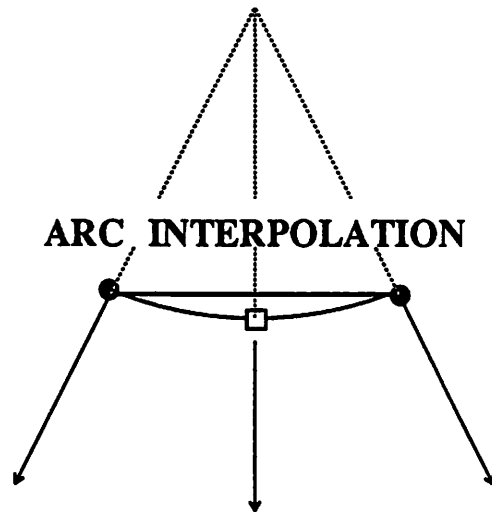
**Figure 7.5 :** 2D etch profiles without and with mesh modification. On the left (a), the rays propagate radially outward, but the segments are not resized. This results in facets. On the right (b), the segments are cut if they become too long, thus producing circular etch profiles.

**LINEAR INTERPOLATION**



(a)

**ARC INTERPOLATION**



(b)

**Figure 7.6 :** Segment cutting with different interpolation schemes. In the left figure (a), a new node and a new vector is added using linear interpolation. On the right, (b), the new node is added on the arc of the circle defined by the two direction vectors of the original segment.

interpolation. In fact, fitting a surface to 3D data-points (which usually involves inverting matrices) can be quite costly in terms of computation time.

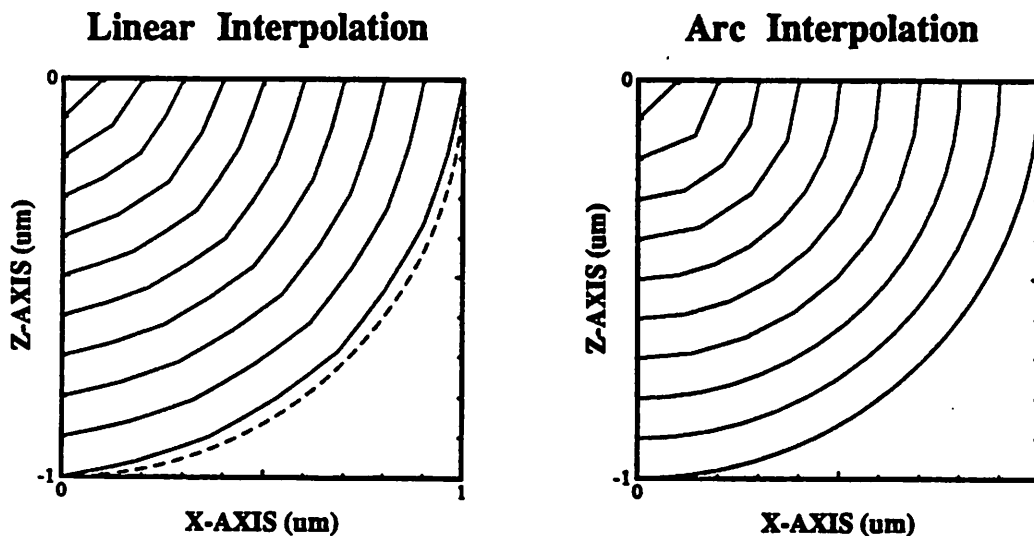
A much easier and faster interpolation method is to add the new node on the arc of the circle defined by the two original direction vectors of the original nodes. Arc interpolation, shown in Figure 7.6b, actually produces better results than the linear interpolation in this particular test case.

Figure 7.7 shows the ray-string simulated results for a uniform circular etch. The simulation begins with a single short segment placed at the upper left corner of a square. The two nodes on the segment have ray vectors initially oriented vertically and horizontally respectively. The simulation proceeds for 10 time-steps of 0.1 seconds each. After every time-step, the segment length of each segment is checked against the ideal segment length. If the segment length  $S$  is greater than the maximum allowable length  $S_{\max}$ , then the segment is cut up. (For this particular simulation,  $S_{\max} = 1.2S_{ideal}$ .) From Figure 7.7, it is easily seen the arc interpolation produces more accurate results than linear interpolation; the profile after 10 time-steps lies exactly on the dashed semi-circle. This is true for both large and small ideal segment lengths. It is also seen that when linear interpolation is used to add nodes to the mesh, the use of smaller ideal segment lengths results in smoother and more accurate profiles. But note that the linearly-interpolated small-segment simulation is still not as accurate as either of the simulations with arc interpolation.

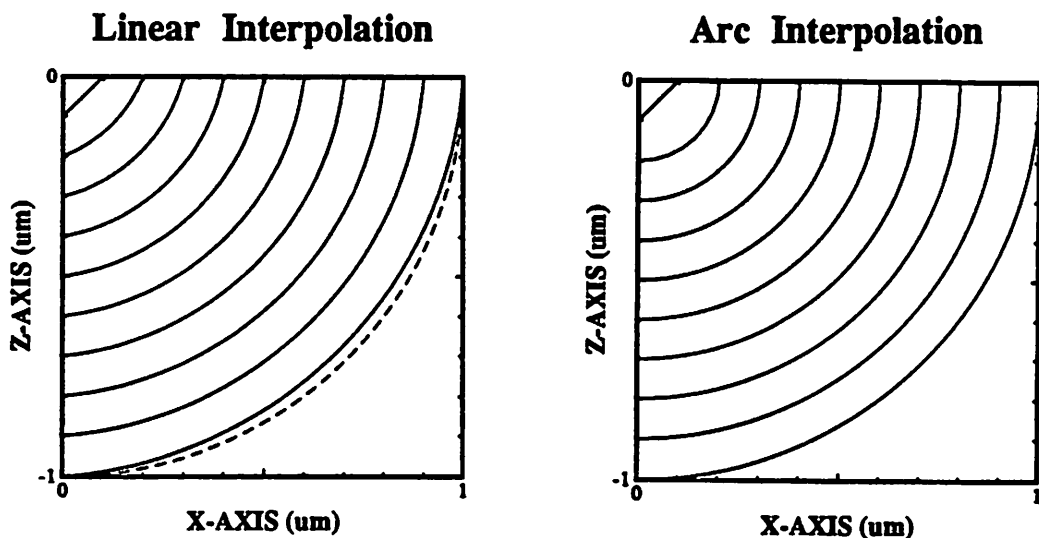
#### 7.5.1.2. Segment Merging

When nodes on a mesh move too closely together, portions of the mesh will become too densely packed. This is inefficient in a computational sense, because nodes that are placed

## IDEAL SEGMENT LENGTH = 0.10 $\mu\text{m}$



## IDEAL SEGMENT LENGTH = 0.02 $\mu\text{m}$



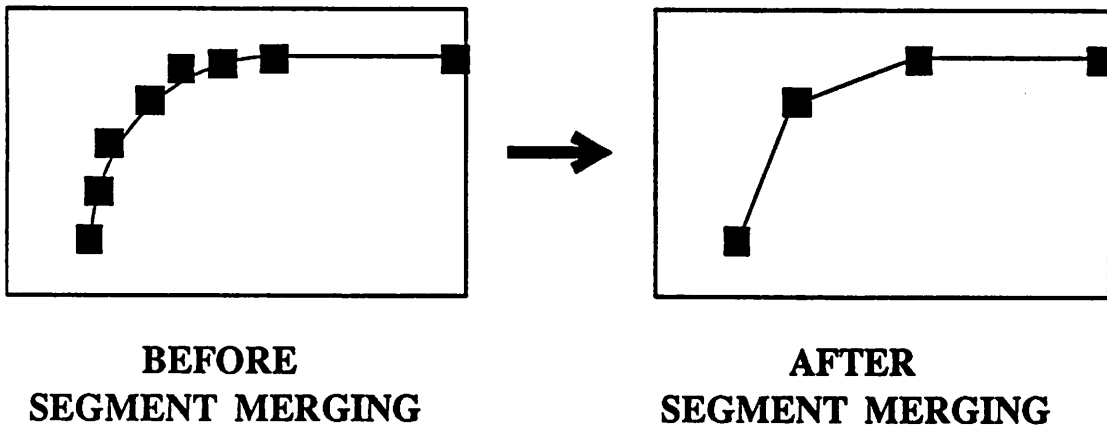
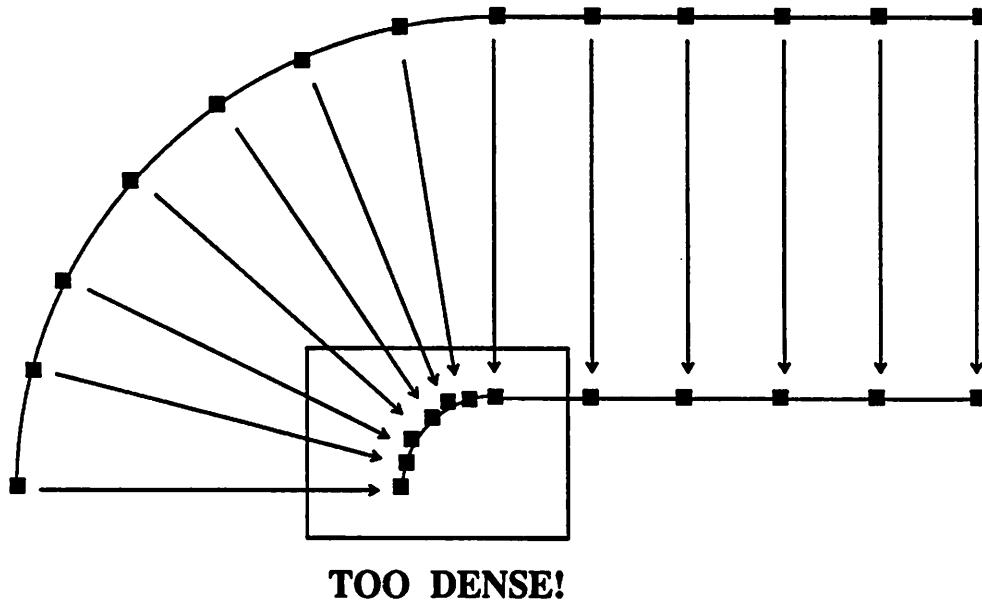
**Figure 7.7:** 2D simulation of uniform circular etching with the ray-string algorithm, using linear and arc interpolation with two different ideal segment lengths. The etch begins from a single segment placed at coordinates (0,0), and proceeds for 1 second of etching time. The time-step is 0.1 seconds. The dashed line shows the expected semi-circular result.

close together essentially bear the same coordinate and vector information. For example, in Figure 7.8, nodes on a circular front have converged. Consequently, the curved portion of the mesh is dense compared to the straight portion of the mesh. There is nothing really wrong about this; in fact, the dense circular mesh is quite accurate in its description of the etch surface. But to be efficient, there should be fewer nodes on the curved portion of the mesh. Fewer nodes means fewer computations, which translates into faster computation speed. If, however, no nodes are deleted in dense regions while nodes are added continually in sparse regions of the mesh, then the number of nodes on the mesh will only grow larger, thus increasing the computation time. Also, if the number of nodes in the mesh is allowed to grow unchecked, mesh operations such as delooping and clipping will also become more computationally expensive. Another consideration is that a dense mesh is ideal for loop formation; if a dense mesh suddenly encounters a high etch-rate region, loops could be formed!

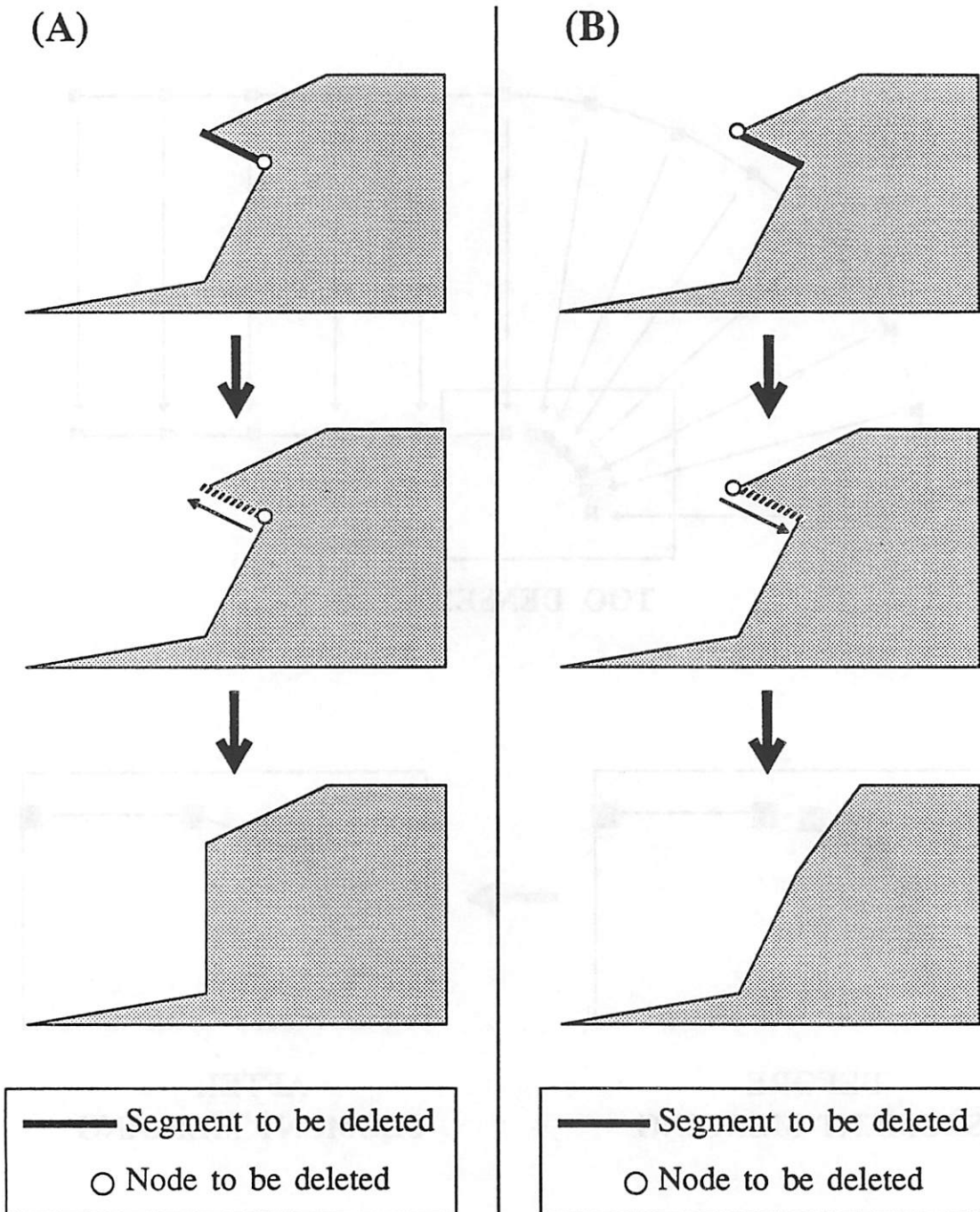
Therefore, to speed up the computation, nodes should be deleted in dense regions on the mesh. An example of node deletion is shown in the lower portion of Figure 7.8. The six segments on the curved surface are merged to produce two longer segments. There is some loss of accuracy, of course, but the tradeoff is that the computation time is decreased and the efficiency of the simulation is increased.

Some care must be taken in the selection of the minimum allowable segment length  $S_{\min}$ . In the segment merging procedure, segments shorter than this minimum ( $S < S_{\min}$ ) will be deleted or merged. But if the segments merged are too long, then too much detail will be lost and the profiles will lose accuracy. An example of this is shown in Figure 7.9 for a 2D profile. In this case, the short segment to be deleted is located on a sharp corner. If this not-so-short segment is deemed too short, its deletion will result in a rounded profile quite dissimilar to the original profile. The merged profile is also sensitive to the choice of node to be





**Figure 7.8:** Nodes on a mesh could move closer together, forming a densely packed mesh. This is accurate, yet computationally inefficient. Less accurate but more efficient results can be obtained by merging short segments.



**Figure 7.9:** 2D Segment merging. Short segments that are deleted could result in inaccurate profiles. The way in which nodes are merged will also affect the final profile.

merged. If a different node on the segment is merged, as in Figure 7.9b, the profile becomes markedly different from that in Figure 7.9a. In the next chapter, it shall be shown that this non-commutative node-merging behavior will produce asymmetrical 3D profiles.

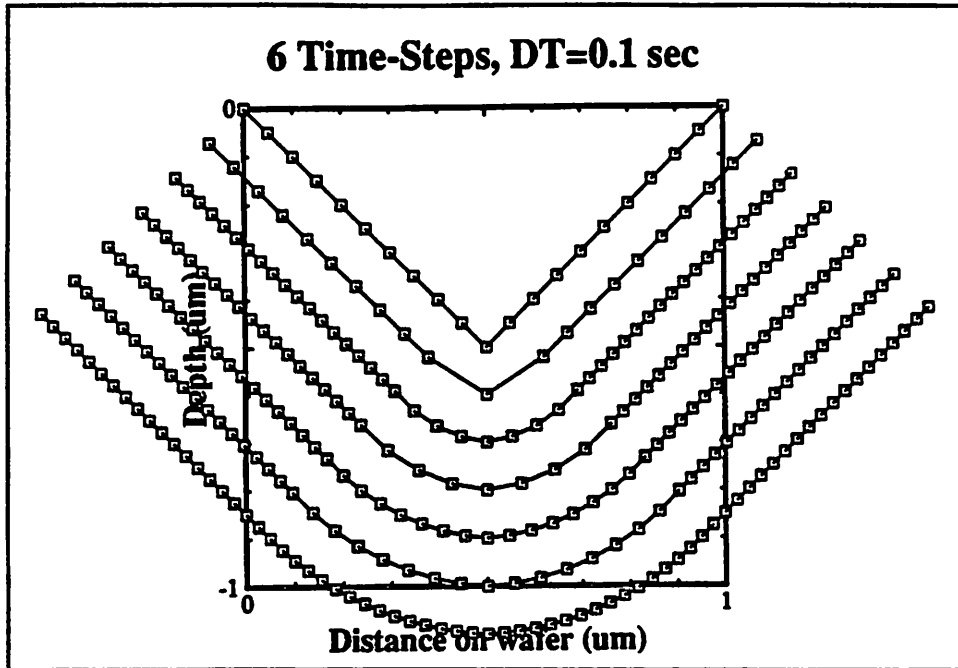
## 7.5.2. Mesh Clipping and Other Boundary Operations

Surface-advancement algorithms also have to be able to handle the problems encountered at the simulation boundaries. The ray-string algorithm is no exception to this.

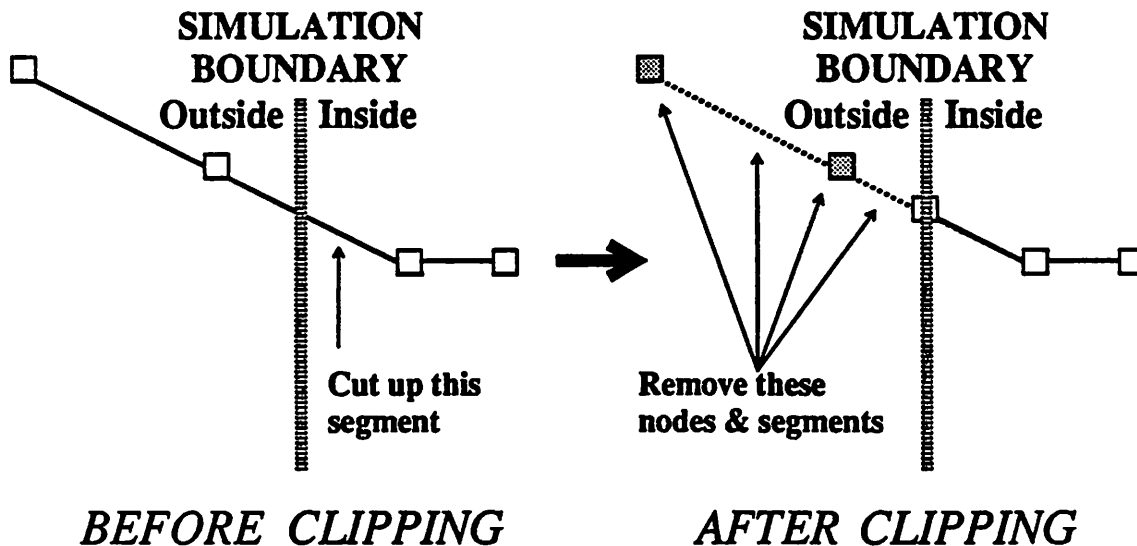
### 7.5.2.1. Mesh Clipping

The typical etching simulation aspires to determine the time-evolving surface inside a rectangular box, where the etch-rate distribution has been defined. The etch-rate is often not known outside the box, so the surface profile outside the boundaries of the box is of little interest. However, as the surface mesh moves and evolves in time, portions of the mesh could move outside the simulation boundaries. These portions of the mesh should be clipped because they do not contribute to the solution inside the surface.

Figure 7.10 is a good example of a mesh that has grown outside the simulation boundaries. The simulation begins with a 90° grooved surface, and the profile etches outward at a uniform rate. After a few time-steps, most of the material inside the original square has been etched away, and large portions of the 2D mesh are outside the simulation boundary. These portions of the mesh are not of interest; only those points of the mesh inside the box are of use in the simulation. Consequently, the continual tracking of the points outside the simulation boundaries is a waste of both computer time and memory. Instead, these points should be deleted.



**Figure 7.10 :** A surface mesh could move outside its simulation boundaries. In this case, the simulation begins from a grooved surface, and proceeds outward at a uniform rate.



**Figure 7.11 :** 2D Clipping : First find the intersection of the segments with the boundary. Cut up the segments that intersect the boundary. Then remove all the nodes and segments outside the boundary.

In two dimensions, clipping involves searching for those segments that cross the simulation boundary. The segments that do intersect the boundary are cut up at the boundary intersection. Then, all the nodes and segments outside the boundary are deleted. This process is illustrated in Figure 7.11. Figure 7.12 shows the results of clipping applied to the uniform grooved etch discussed above. In the simulation, the mesh was clipped after every surface advancement. In practice, though, it is not desirable to clip after every time-step, since clipping is an operation expensive in terms of computation time. A more efficient approach is to clip every 10 time-steps or so.

An alternative to clipping is to stop the nodes from moving outside the boundary. One way to do this is to set the etch-rate to zero outside the simulation boundaries.† The etch profiles simulated under this scheme are shown in Figure 7.13. This method is easier than clipping, primarily because it is no longer necessary to search for boundary intersections. However, this method does have drawbacks. One of these is that excess nodes and segments will accumulate along the simulation borders. These nodes and segments do not contribute to the simulation, and as a result, computation time and memory is wasted. Another drawback is that the profiles can have lagging tails at the boundaries. As shown in the illustration on the left of Figure 7.13, a lagging tail will be formed if one node of a segment is pinned to the boundary while the other node is left free to move inside the boundary.

---

† However, as discussed in relation to Figure 6.17 (Section 6.3.3.3), the differential ray equation does not hold at abrupt rate-boundaries, where the etch-rate is discontinuous. Therefore, the ray equation cannot be used in a scheme where the etch-rate suddenly drops to zero. To get around this, the etch-rate can be gradually ramped down to zero in a narrow region outside the boundaries. In the simulation of Figure 7.13, the etch-rate is linearly ramped down to zero in an intermediate region of thickness  $0.01 \mu\text{m}$ .

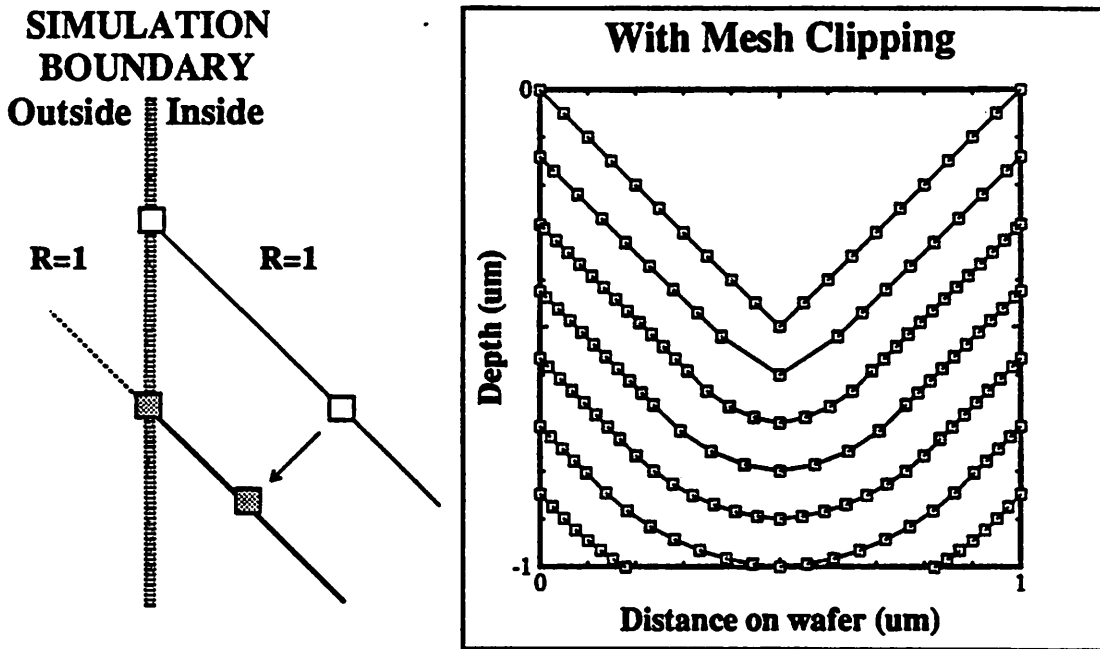


Figure 7.12 : Uniform etch from a grooved surface, with clipping after every surface advancement.

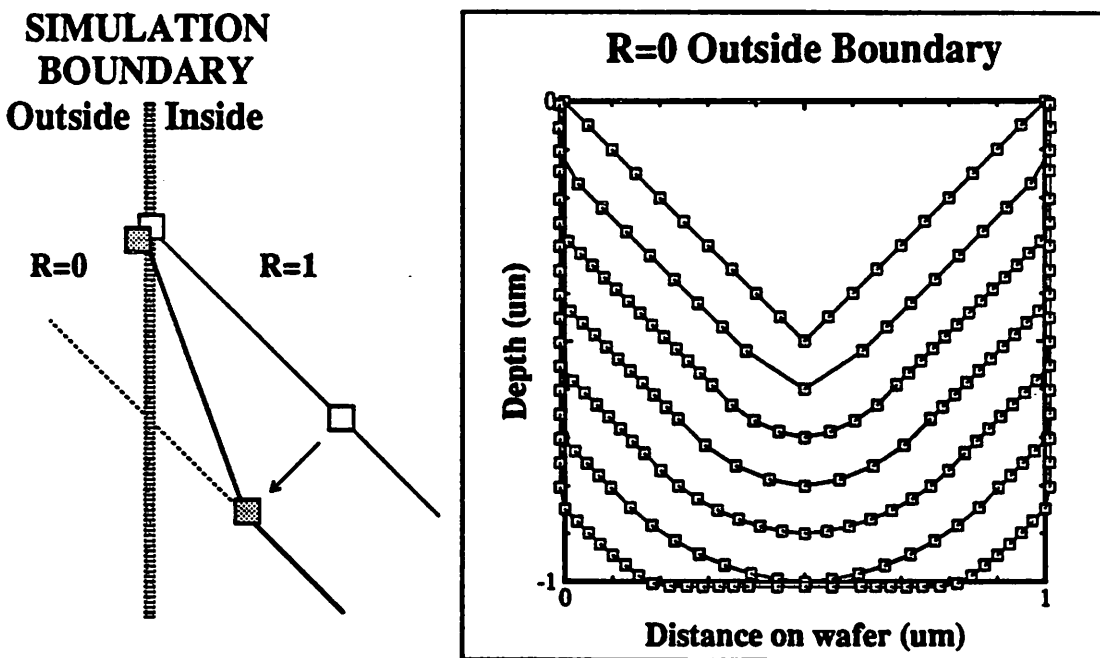


Figure 7.13 : Zero-etch-rate outside the boundaries results in the above profiles. This method, however, could produce lagging profiles, as illustrated on the left. Note that the second profile in the simulation has developed a lagging tail at the x-boundaries.

### 7.5.2.2. Boundary Flattening

The inverse of mesh expansion is mesh contraction. During the course of the simulation, the mesh could also contract; some nodes on the boundary could move inside the borders. Mesh contraction occurs quite commonly during surface advancement. If a node on the boundary has a direction vector that points into the simulation volume, then that node could move away from the boundary. This situation is illustrated in Figure 7.14a. If the node is allowed to move away from the boundary, part of the true etched surface will no longer be blanketed by a mesh. As a result, the simulation is no longer correct.

An example of a simulation where mesh contraction occurs is shown in Figure 7.14b. The simulation is run with a triangular etch-rate

$$R(x,y,z) = 2|x| \mu\text{m}/\text{sec} \quad [7.8]$$

for 10 time-steps of 0.1 second duration each. As can be seen, the etch-fronts curve inwards towards the  $x = 0$  plane. This behavior was also observed in the ray-tracing simulation shown in Figure 6.13a (Section 6.3.3.2), where the trajectories of the individual rays were calculated. It is not difficult to see that the rays in Figure 6.13a and the etch-fronts in Figure 7.14b are orthogonal to each other. And as in Figure 6.13a, there are also ray-scarce regions in the ray-string simulation.

In order to avoid the incomplete-mesh situation just described above, it is necessary to pin the boundary nodes to the boundary. So, in Figure 7.15a, the direction vector of the node on the boundary should be adjusted so that the vector is parallel to the boundary. To do this, the vector must be "flattened"; the component of the vector normal to the boundary surface is set to zero, and the vector is renormalized. As shown in Figure 7.15a, the result of the vector "flattening" procedure is that the node on the boundary will slide along the boundary, and the

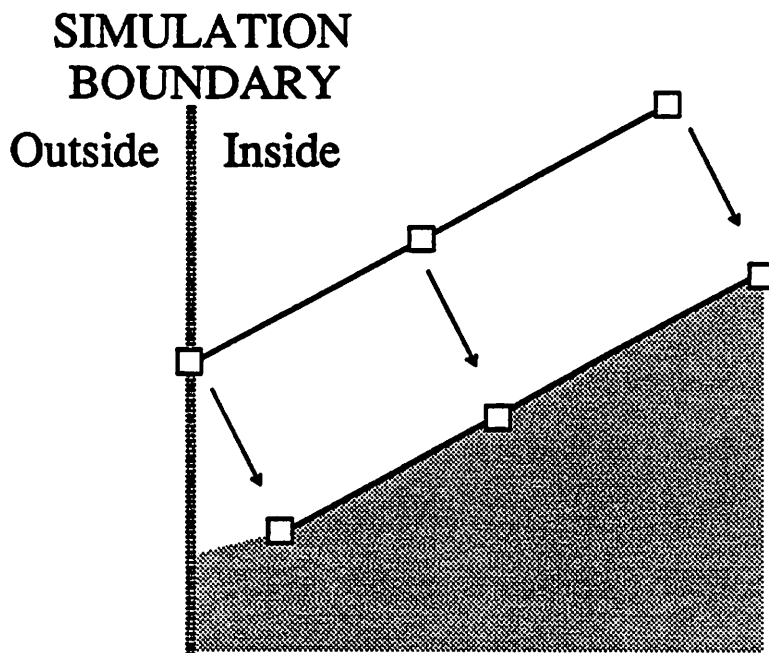


Figure 7.14a : 2D Boundary Movement. The nodes on the boundary could move inside the boundary, leaving part of the surface uncovered by the mesh.

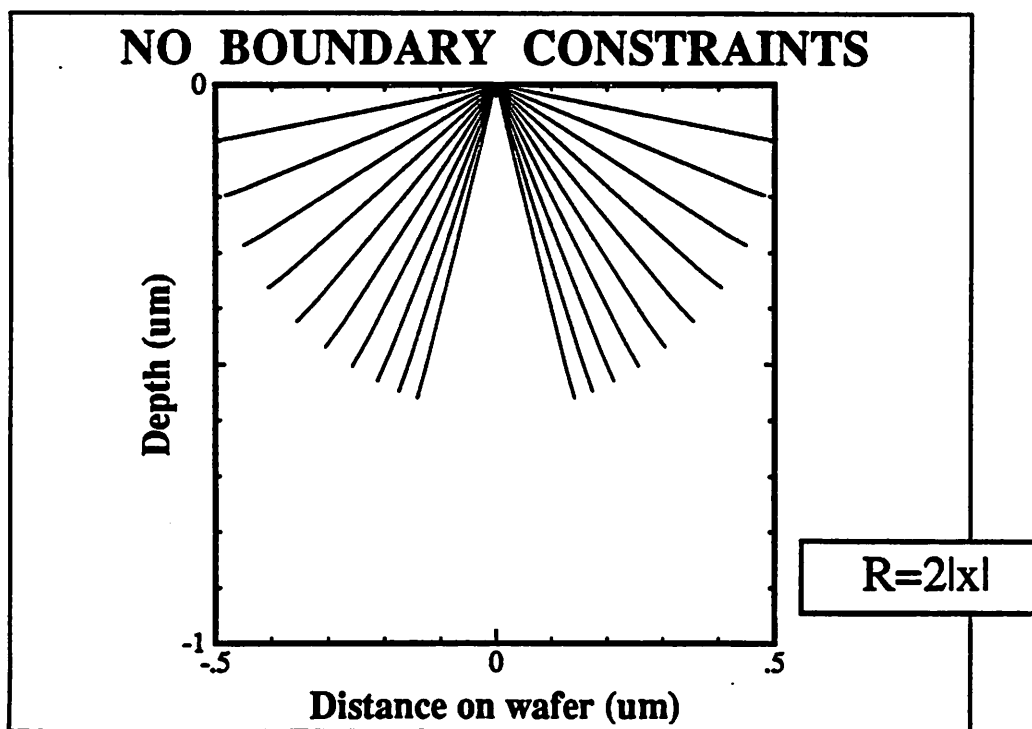
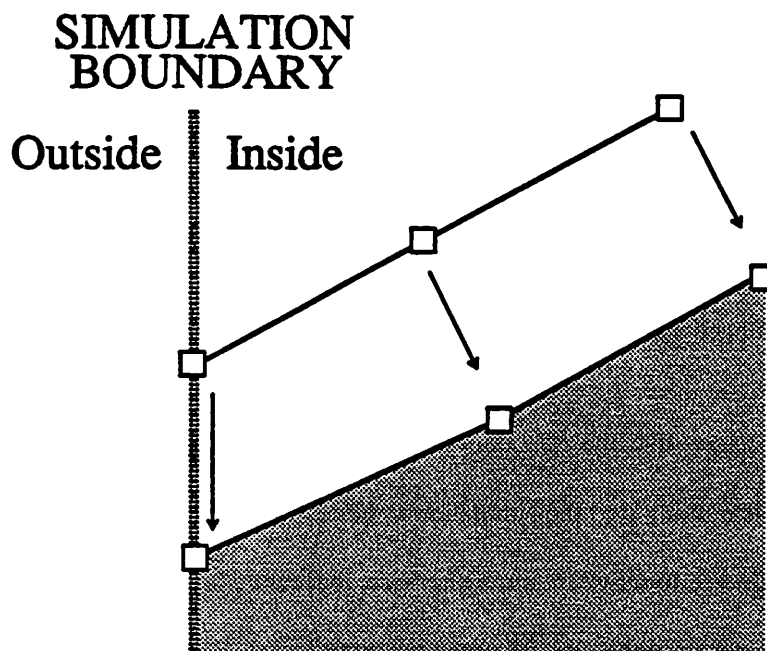
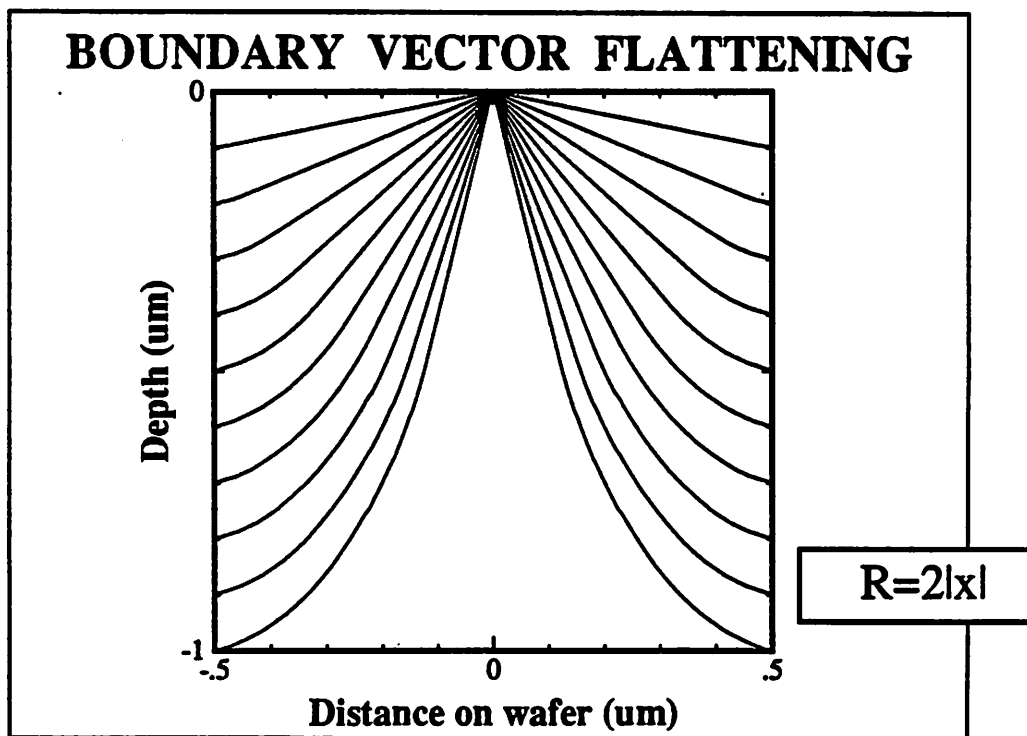


Figure 7.14b : 2D ray-string etching using the triangular etch-rate function  $R=2/|x|$ . The simulation is run without any boundary constraints, so that the border nodes can and do leave the border. The simulations were run for 10 time-steps of 0.1 seconds each.





**Figure 7.15a :** Nodes on the boundary should only be allowed to move on the boundary. The node's direction vector is "flattened" so that the node can only move on the boundary.



**Figure 7.15b :** 2D ray-string etching using the triangular etch-rate function  $R=2|x|$ . In the simulation, nodes on the boundary are prohibited from moving into the simulation volume. The simulations were run for 10 time-steps of 0.1 seconds each.

full etch surface will remain covered by the mesh. Figure 7.15b shows the profiles of the triangular etch-rate simulation with vector flattening. The profiles formed now cover the entire etch surface; the mesh has been prevented from contracting inwards. Note that these profiles also look very similar to those shown in Figure 4.4b, where the modified cell algorithm was used to determine the etched surface.

It must be emphasized that the boundary-node vectors should be flattened only if the node on the boundary is moving into the simulation volume. If the node is moving out of the volume, it should be allowed to move freely. Otherwise, inaccurate lagging tails such as those seen in Figure 7.13 will occur in the vicinity of the boundary.

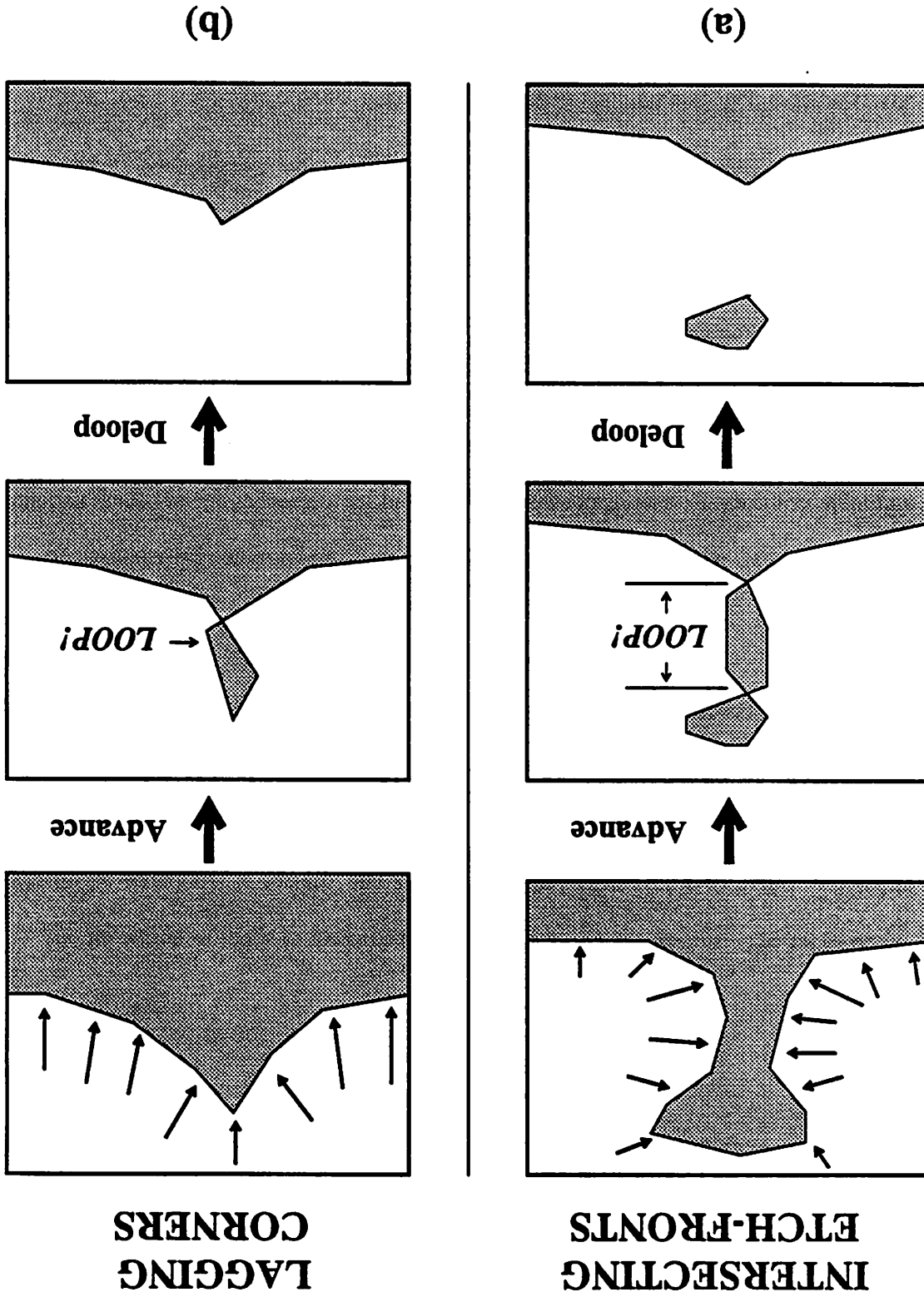
### **7.5.3. Mesh Delooping**

#### **7.5.3.1. Loop Formation**

Loop formation is perhaps the greatest weakness of the surface-advancement algorithm. Surface advancement algorithms move the mesh representing the etch surface without foreknowledge of the etch-state in the local volume of the material. As a result, the mesh could move into an already-etched region and so form loops.

In general, loops will be formed at the intersection of two converging etch-fronts. Examples of loop formation are shown in Figure 7.16. In the example in Figure 7.16a, the surface is being etched both from the left and from the right; physically, the etchant is tunneling through the tall mesa-like structure to form a bridge. The loop is formed when the etch-fronts converge. The removal of the loop leaves a blob of material floating freely - this is actually the 2D cross-section of a bridge.

Figure 7.16 : Loops could be formed at lagging corners or at the intersection of converging etch-fronts. The mesh then has to be delooped, that is, the loops must be removed.



Loops could also be formed at sharp converging corners, as seen in Figure 7.16b. If the mesh moves slower at the corner than at the sides of the corner, a loop could be formed when the etch-fronts from the sides of the corner converge. The loop then has to be removed.

### 7.5.3.2. Loop Avoidance

The formation of loops in the surface mesh poses difficult problems. The loops do not represent the actual etched surface, and are present only because the algorithm has failed to keep track of the true etched surface. Unlike the volume etching methods discussed in Chapters 3 and 4, the string, the ray, and the ray-string etching methods do not retain information on the etch-state throughout the volume of the material. As a consequence, the etch-fronts are free to move into already-etched regions.

One way of dealing with this problem is to keep track of the etch-state in the entire volume of the material. Scheckler<sup>2</sup> has proposed a hybrid cell/surface-advancement algorithm in which a cellular array is used to store the etch-state of the material at uniform grid-points throughout the volume of the material. In some ways, this is similar to the cell-removal method, but the decision of cell-removal is made based on the surface motion and thus many cells may be removed at a single time-step. The hybrid cell/surface-advancement algorithm has been demonstrated in preliminary 3D simulations of small-volume photoresist development. However, the algorithm does run into memory allocation problems when the volume of the material to be simulated is large, since large simulation volumes also require large arrays. The algorithm also pays a price in computation time and complexity for the continual updating of the cellular array.

Another method for avoiding loops is to remesh the surface periodically with splines. Barouch<sup>3</sup> uses a tensor-product B-spline interpolation function to locally smooth the evolving surface at every time-step. This method has been demonstrated in photoresist simulations with weak standing waves. Unfortunately, the use of splines to smooth the surface is computationally expensive - not much specific data is available, but the work reported by Barouch was done on a supercomputer. Furthermore, surface smoothing or remeshing can introduce errors, since the surface is unduly affected by a few incorrect points. Another consideration is that surface-smoothing with splines will only work locally; it cannot handle cases in which distant etch-fronts intersect as in Figure 7.16a.

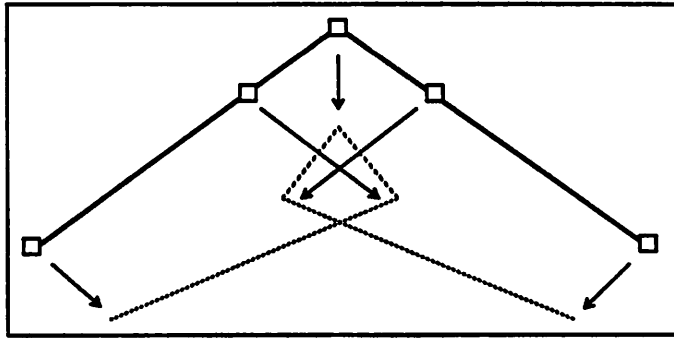
### **7.5.3.3. 2D Delooping**

As indicated in the previous section, avoiding loops is both difficult and computationally expensive. So, if it is not possible to detect and avoid the loops before they do occur, the only recourse is to remove the loops when they do form. But it is not always easy to predict when, where or even if loops will be formed. So, the next best thing is to inspect the surface mesh periodically and remove any loops that are encountered.

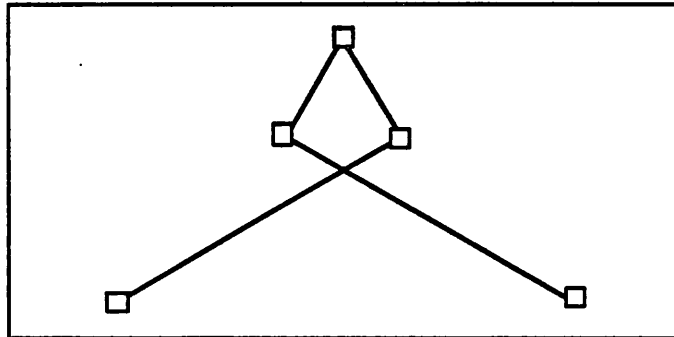
Figure 7.17 shows the procedure for removing loops in 2D. This procedure can be broken up into four steps.

- [I] Find the intersecting segments by examining the list of segments.
- [II] Cut up the segments that do intersect.
- [III] Determine the segments and nodes in the loop.
- [III] Remove the segments and nodes inside the loop.

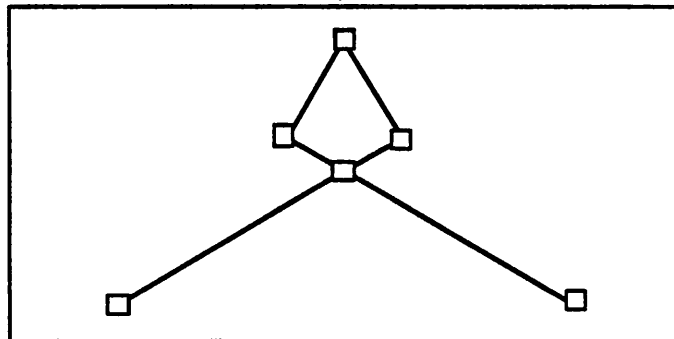
**Loop Formation**



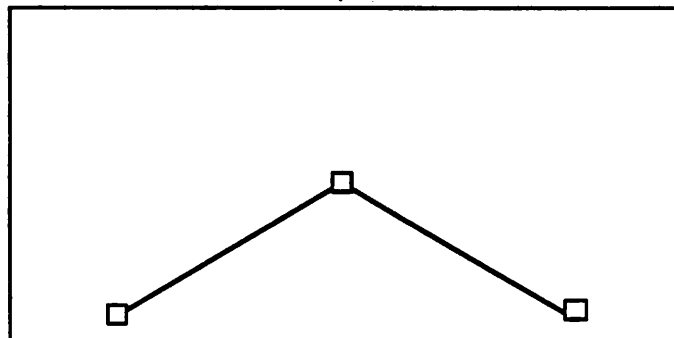
**Find the  
intersecting segments**



**Cut up the  
intersecting segments**



**Remove segments  
in the loop**



**Figure 7.17 :** Procedure for removing loops

Of these four steps, the most difficult is that of determining which of the segments and nodes in the surface mesh are inside or outside the loop. One method of determining which points in the mesh are inside or outside the loop is outlined in Figure 7.18. This method relies on the direction of the motion of the intersecting segments to determine which of the nodes on the segments are inside or outside the loop. In Figure 7.18a, it is easily seen that node  $NB_2$  is behind segment  $SA$ , since the two nodes on  $SA$  are moving away from  $NB_2$ . Similarly, node  $NA_2$  lags segment  $SB$ . Therefore, the nodes  $NA_2$  and  $NB_2$  are both in the loop, while the other two nodes,  $NA_1$  and  $NB_1$  are outside the loop. The other nodes in the loop can now be identified by traversing the loop, that is, by moving along the segments adjoining the loop nodes  $NA_2$  and  $NB_2$ . And once these loop segments and nodes have been identified, they can be removed.

An example of the delooping procedure is shown in Figure 7.19, in which the profiles have been simulated using the ray-string algorithm with the the triangular etch-rate function

$$R(x,y,z) = 2|x| + 0.2 \mu\text{m/sec}, \quad |x| < 0.5 \quad [7.9]$$

As can be seen, the profile develops a loop after 8 time-steps. The loop is removed using the delooping procedure described earlier. However, when the mesh is advanced further, it intersects again at a later time. The intersection of the two fronts again has to be removed. After the final deloop, the profile in the lower left of Figure 7.19 is obtained.

It must be emphasized that the delooping procedure as described above works best when there is only a single loop in the profile. The procedure will not work correctly when the profile is plagued by multiple intersecting loops, such as in Figures 6.7b-c. Figure 7.20a shows another example of a Gordian knot. Clearly, it is not easy to determine which parts of the surface are inside or outside the loop. Another confusing situation arises when one or both of a pair of intersecting segments is a pinwheeling segment. A pinwheeling segment, as shown in

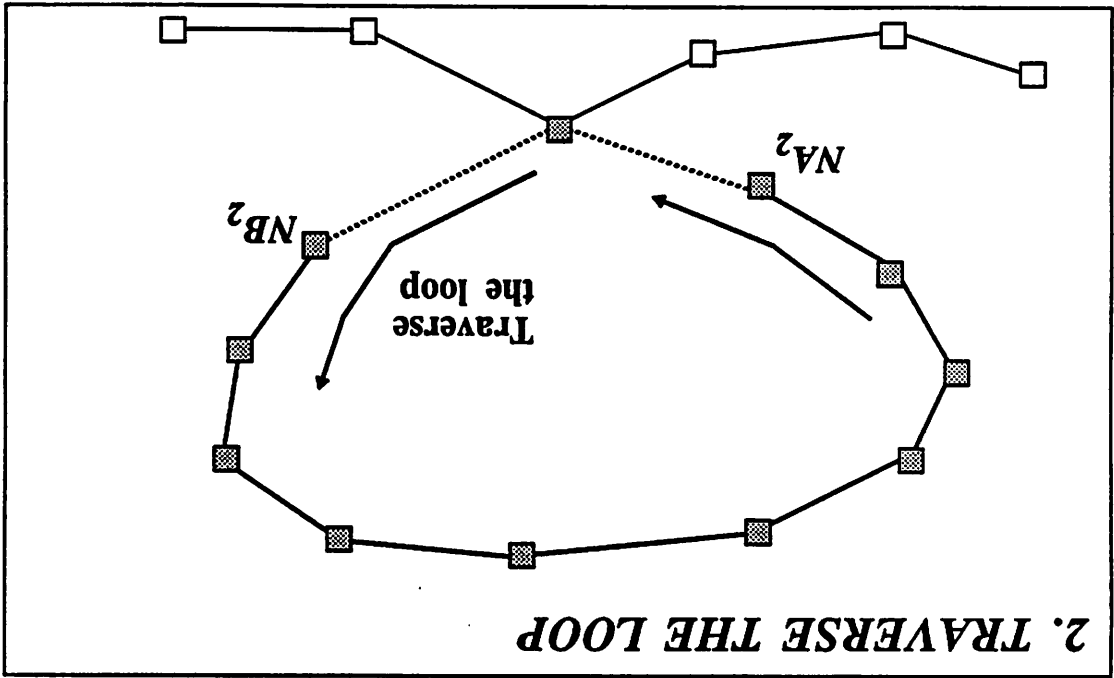
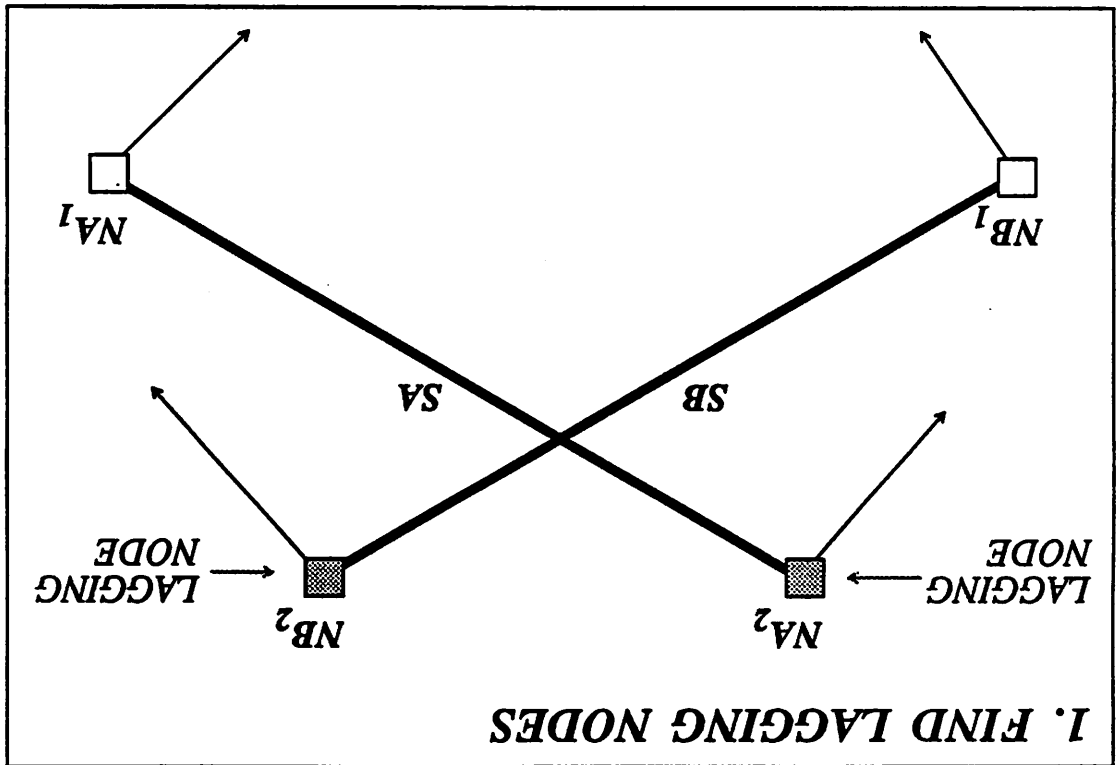
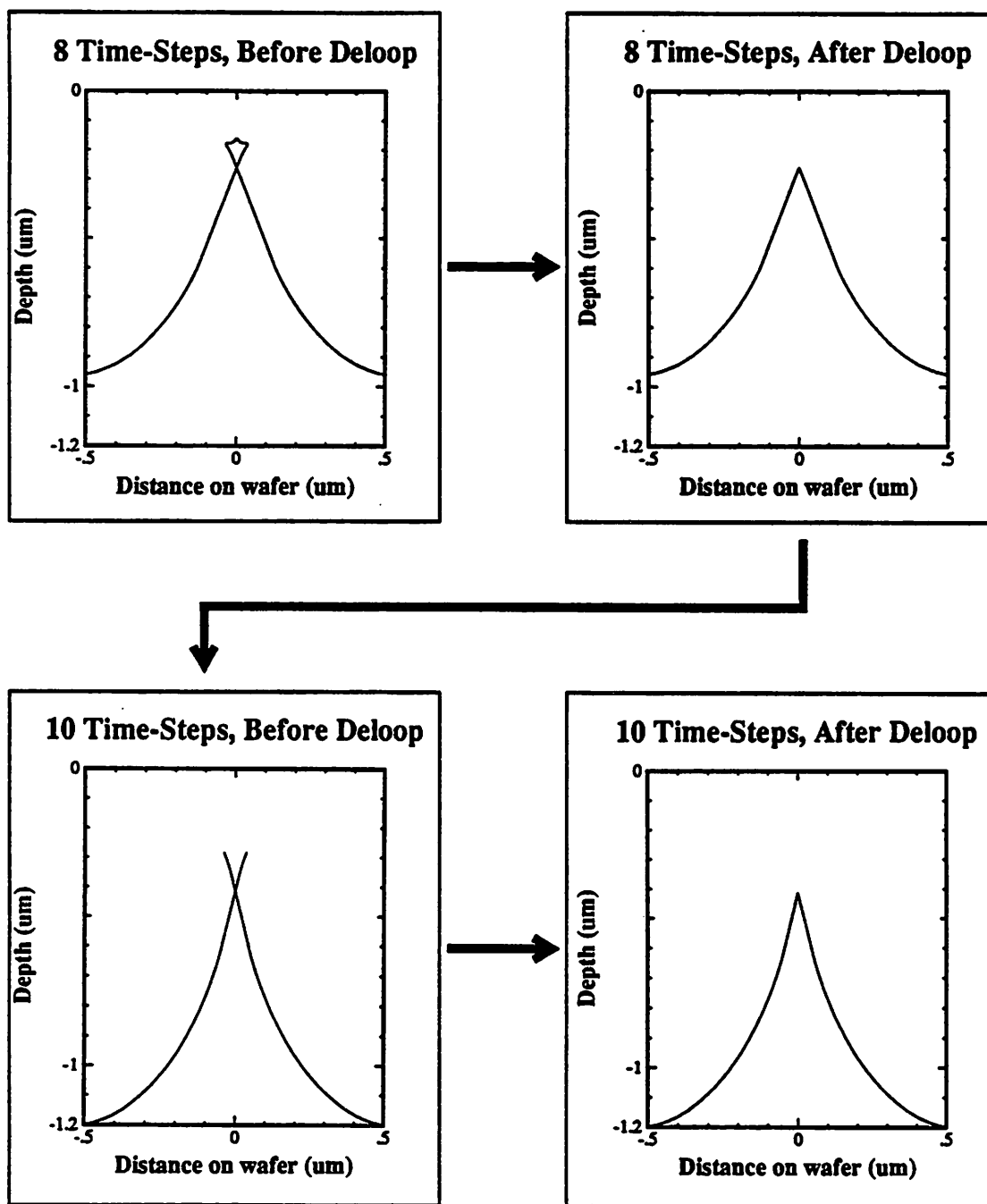


Figure 7.18 : Procedure for finding the segments in the loop. First, find the lagging nodes in the original pair of intersecting segments. Then traverse the loop and identify all the segments and nodes in the loop.





**Figure 7.19:** 2D ray-string etching using the triangular etch-rate function  $R=2/x+0.2$ . The simulations were run for a total of 10 time-steps of 0.1 seconds each. The mesh was delooped at 0.8 seconds and 1.0 seconds.

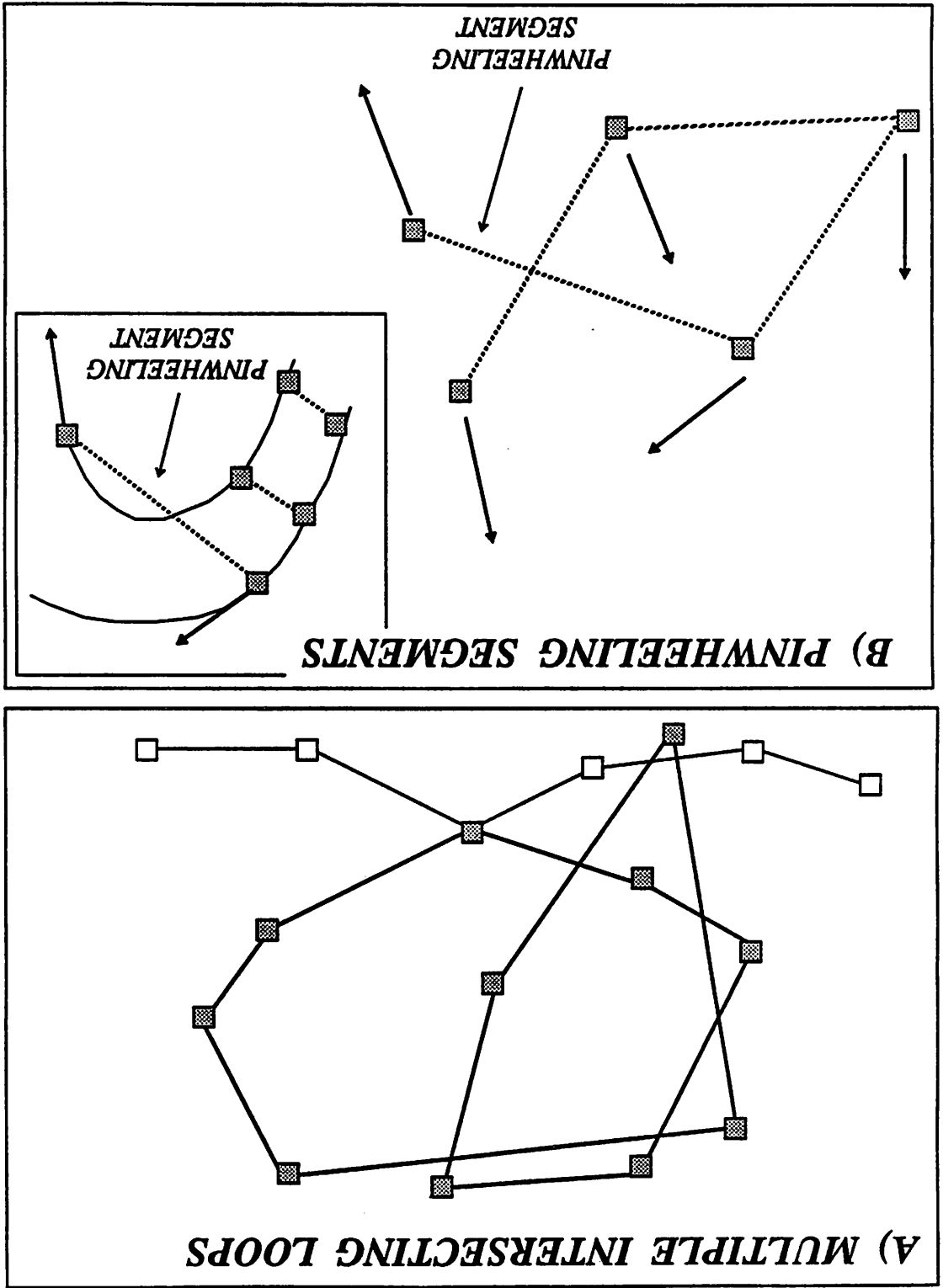


Figure 7.20 : Loop Complications. (a) Multiple Intersecting Loops are quite difficult to unravel. (b) Pinwheeling Segments could cause errors in identifying nodes inside the loop.

Figure 7.20b, is a segment whose nodes are traveling in opposite directions. If an intersecting segment is pinwheeling, it is not easy to determine which part of the surface is etched or unetched. An incorrect determination could lead to the deletion of "correct" nodes outside the loop.

#### 7.5.4. Loop Formation In Photoresist Development

The loop behavior in the ray-string algorithm may be understood better by testing the algorithm with a high-frequency sinusoidal etch-rate function. This function resembles the etch-rate distribution found in photoresist development.

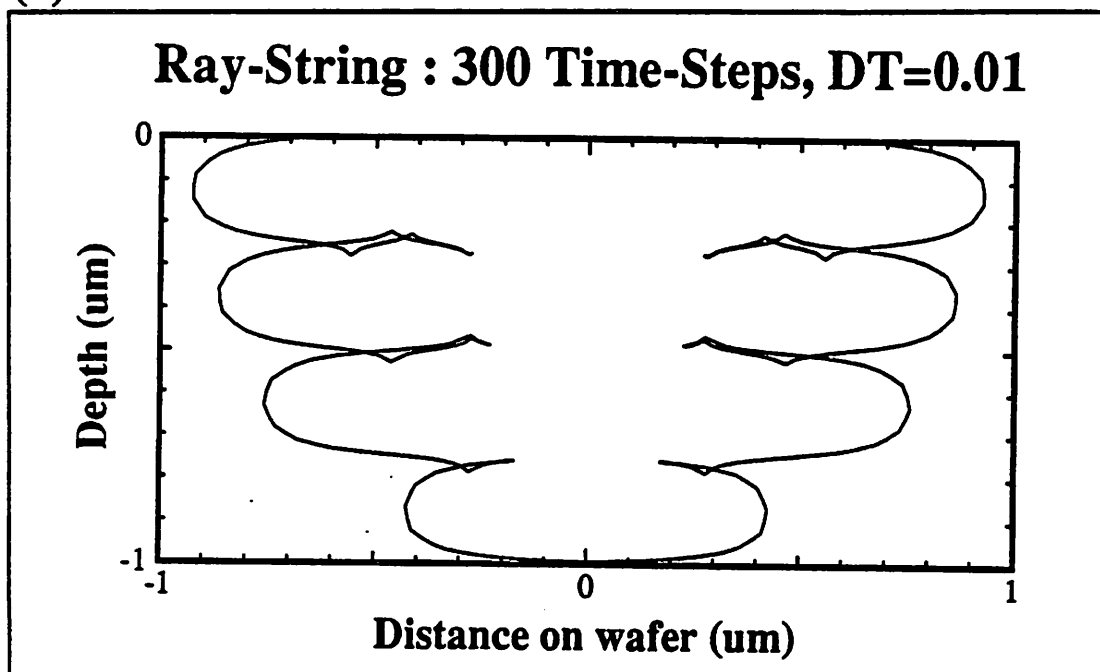
$$R(x, y, z) = e^{-4x^2}(1.05 - \cos(8\pi z)) \quad [7.10]$$

Figure 7.21a shows the etch profile after 300 time-steps. The profile was simulated using the 2D ray-string algorithm with the etch-rate function above. The simulation was run without delooping and with linear (instead of arc) interpolation. It is clearly seen that loops have formed at the corners of the profile where the etch-rate is relatively slow. But also note that the loops are quite narrow and restricted to certain regions in space. In fact, if the ray-trajectories from Figure 6.15b are superimposed on the ray-string profile of 7.21a, as has been done in Figure 7.21b, one can see that most of the multiply-curved rays have ended up on the loops of the ray-string profile! So, in effect, the loops have formed because the rays are trapped and guided in the regions of relatively low etch-rate within the material, very much like optical rays are guided in waveguides.

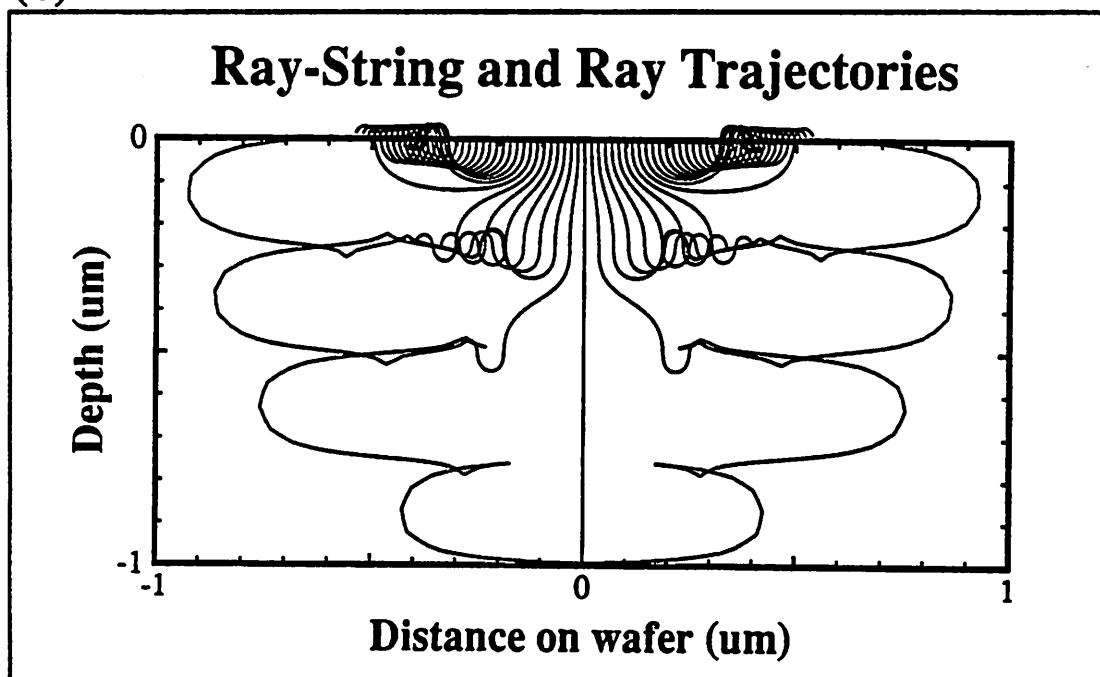
Similar loop behavior is observed when the ray-string simulation is run with real photoresist parameters. Figure 7.22 shows the resist profiles at development times of 3 - 30 seconds. The etch-rate distribution used was that of Figure 2.3, for a 1.25  $\mu\text{m}$  isolated space on 1.0  $\mu\text{m}$  of Kodak 820 resist. Again, loops are formed at the nodes of the standing waves.

$$R = \exp(-4x^2) \cdot (1.05 - \cos(8\pi z))$$

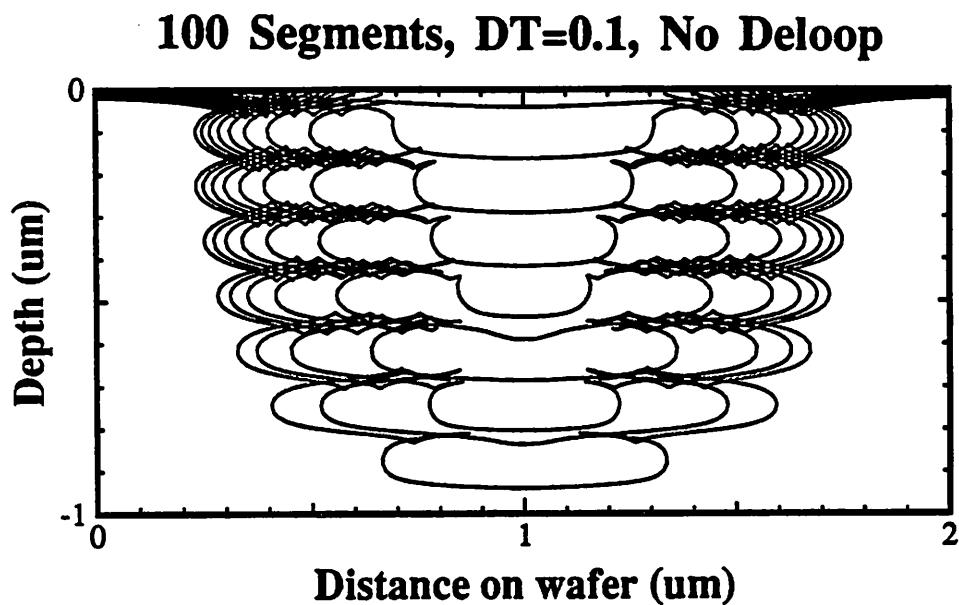
(a)



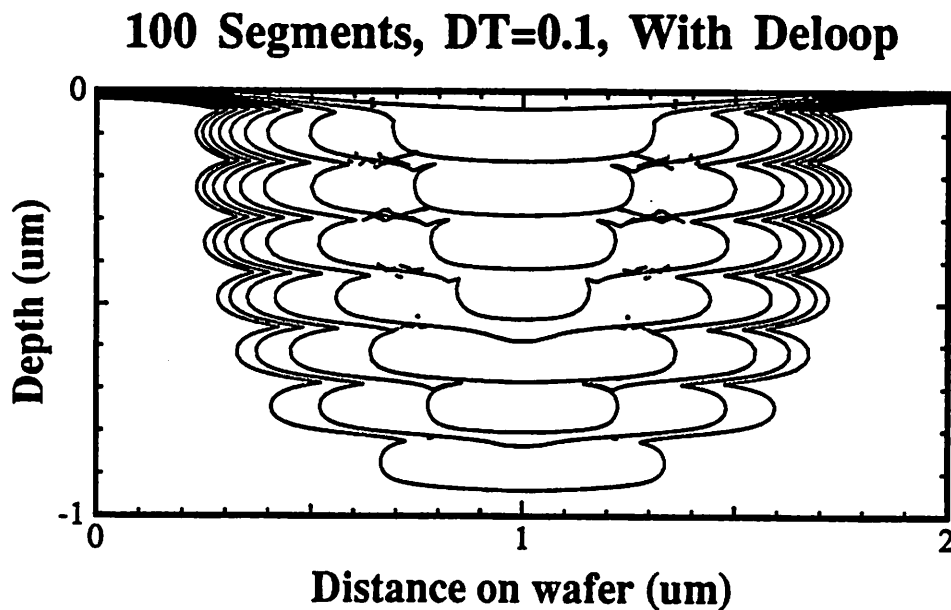
(b)



**Figure 7.21:** 2D ray-string etching using a sinusoidal etch-rate function. Simulation proceeds for a total of 3 seconds, with 300 time-steps of 0.01 seconds each. The ray trajectories from Figure 6.15b are superimposed on the ray-string profile of (a) in the lower figure (b).



**Figure 7.22a :** Resist profile at development times of 3-30 seconds. The etch-rate distribution of Figure 2.3 is used for the ray-string simulation. The mesh has not been delooped.



**Figure 7.22b :** Resist profile at development times of 3-30 seconds. The etch-rate distribution of Figure 2.3 is used for the ray-string simulation. The mesh is delooped every 3 seconds.

And again, the loops are restricted to the low-etch-rate regions.

At this point, it is instructive to compare the behavior of the loops in the string and in the ray-string. Recall that under the same simulation conditions described above, the string simulation produces lots of interlocking loops (Figure 6.7). In contrast, as previously noted, loops formed by the ray-string algorithm are restricted to the low etch-rate regions in the photoresist. This key difference is, as discussed in the previous chapter, due to the way in which the advancement vectors are calculated in the two algorithms.

In order to produce more pleasing profiles, the surface mesh can delooped. In the simulation shown in Figure 7.22b, the profiles have been delooped every 3 seconds (30 time-steps). The profiles become much easier to analyze. But also note that in Figure 7.22b, there are a number of "hanging" segments unconnected to the main body of the mesh. This is a result of errors in the delooping code. It all goes to show that even in 2D, delooping is not easy.

It should also be noted that in theory, it should be possible to use only a single deloop operation at the final step to remove all the loops that have formed in the surface. This is because the ray method of vector calculation restricts the loops to certain regions in space. Unfortunately, final-step delooping is not easy to do in practice. For example, if the simulation described in Figure 7.22 is allowed to run for the full 300 time-steps before delooping, some of the intersecting segments will be pinwheeling segments. These segments are not accounted for correctly as yet, and as a result, the deloop will fail and the program will crash.

## **7.6. TUNING SIMULATION PARAMETERS FOR ACCURACY**

The discussion thus far has shown that the ray-string etch simulator, like any other numerical program, has a number of internal simulation parameters that affect both the accuracy of the results and the efficiency/speed of the simulations. It is quite important to know which parameters are of the most significance. It is also necessary to understand the tradeoffs in efficiency and accuracy obtained from tuning these simulation parameters.

The effect of a number of simulation parameters on the developed resist profile is shown in Figures 7.23 and 7.24. In all cases, the simulations were run using the 2D ray-string algorithm with the etch-rate distribution of Figure 2.3. Each simulation was run for a total of 30 seconds of development time. The effect of the parameters on the simulation is tabulated in Table 7.2.

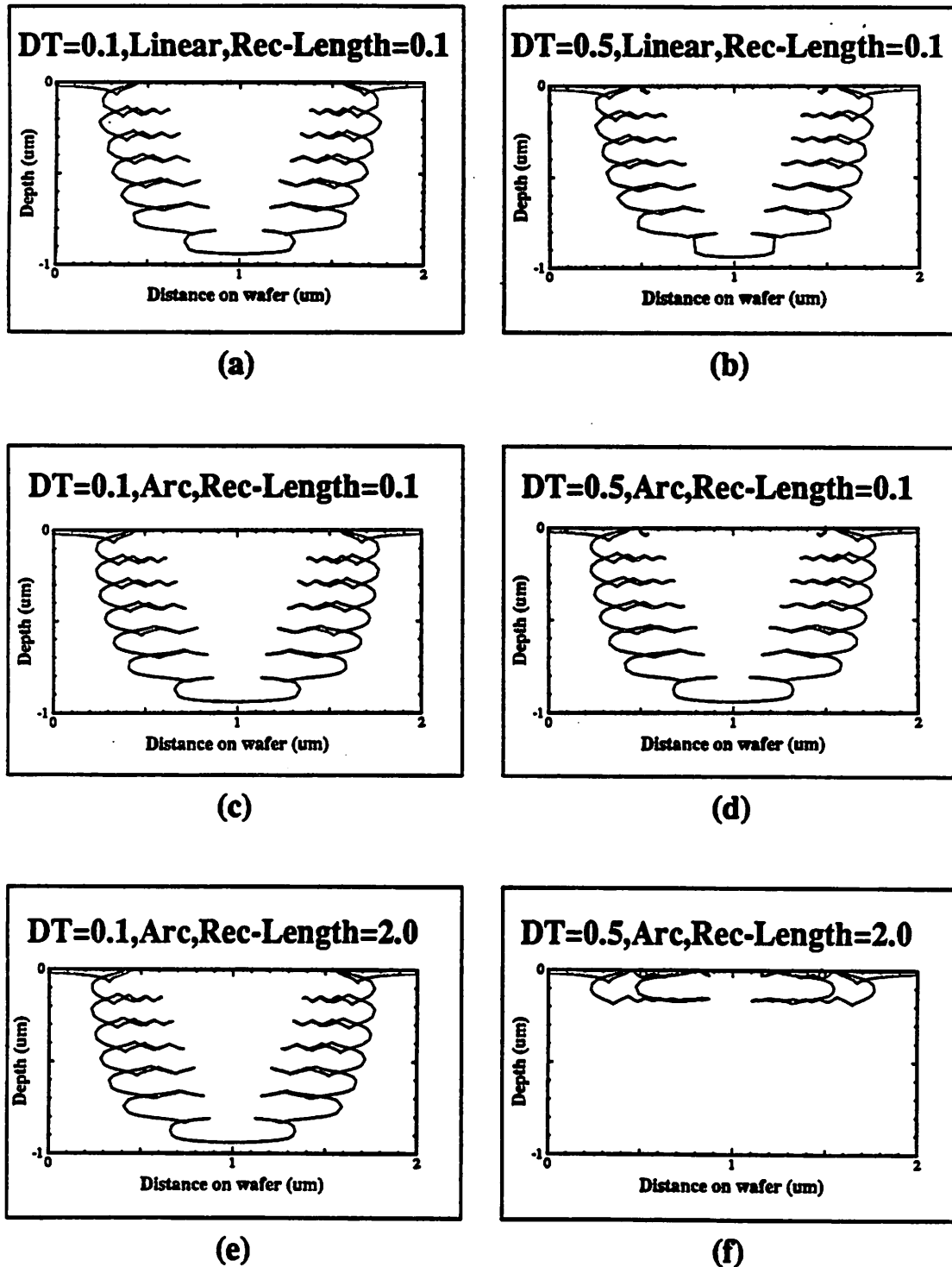
The parameters investigated and their effects are listed below in no particular order.

### **Interpolation Scheme**

There are two interpolation schemes for adding nodes to sparse regions of the mesh. As discussed in Section 7.5.1.1, the nodes may be added on a straight or linear line joining two distant nodes, or they may be added on the arc of a circle defined by the direction vectors of the two distant nodes. The profiles produced by arc interpolation (Fig 7.23c,d) are definitely smoother than those produced using linear interpolation (Fig 7.23a,b). The computation time does not appear to change significantly when either of the two methods are used.

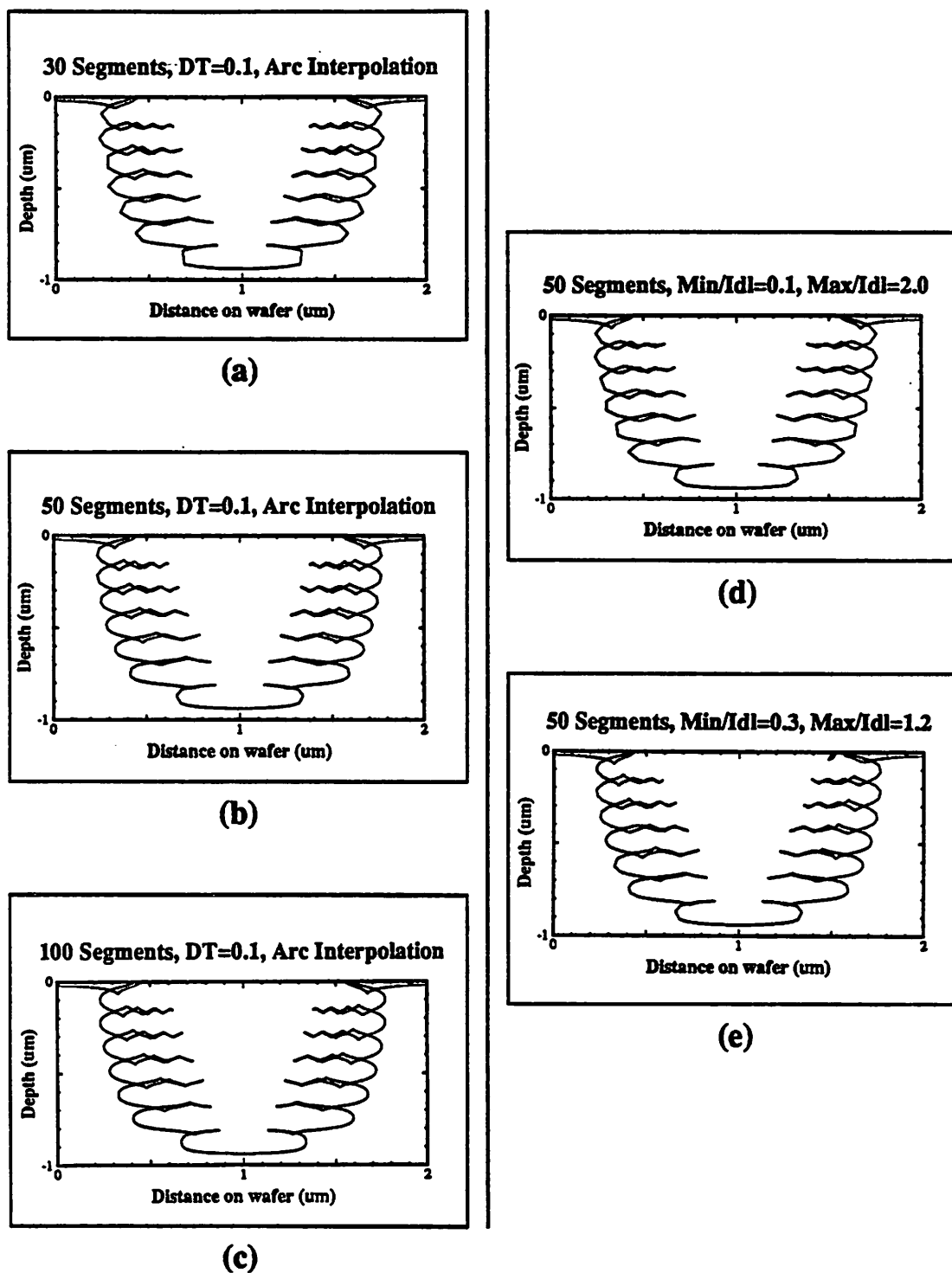
### **Time-Step Size**

The size of the time-step is quite important in determining the accuracy of the simulations. As might be expected, the results do become more accurate when a smaller time-



**Figure 7.23 :** The effect of the interpolation scheme, time-step size, and recursive length on the simulated resist profile after 30 seconds of development. The segment length and allowable range are kept constant. The etch-rate distribution of Figure 2.3 is used for the ray-string simulation.





**Figure 7.24 :** The effect of the ideal, maximum and minimum segment lengths on the simulated resist profile after 30 seconds of development. All the simulations use arc interpolation with time-step of 0.1 seconds and recursive length of 0.1  $\mu\text{m}$ . The etch-rate distribution of Figure 2.3 is used for the ray-string simulation.

Effect of Simulation Parameters on 2D Ray-String Simulation							
Constant Segment Length (Figure 7.23)							
Figure	Segments	Interpolation Scheme	Time-Step Size	Recursive Length	Max/Min Segm Size <sup>1</sup>	Surface Smoothness	CPU Time <sup>2</sup>
7.23a	50	Linear	0.1	0.1	1.2/0.1	Moderate	80 sec
7.23b	50	Linear	0.5	0.1	1.2/0.1	Rough	54 sec
7.23c	50	Arc	0.1	0.1	1.2/0.1	Smooth	80 sec
7.23d	50	Arc	0.5	0.1	1.2/0.1	Smooth	52 sec
7.23e	50	Arc	0.1	2.0	1.2/0.1	Smooth	67 sec
7.23f	50	Arc	0.5	2.0	1.2/0.1	N/A <sup>3</sup>	10 sec
Variations In Segment Length (Figure 7.24)							
Figure	Segments	Interpolation Scheme	Time-Step Size	Recursive Length	Max/Min Segm Size <sup>1</sup>	Surface Smoothness	CPU Time <sup>2</sup>
7.24a	30	Arc	0.1	0.1	1.2/0.1	Moderate	53 sec
7.24b <sup>4</sup>	50	Arc	0.1	0.1	1.2/0.1	Smooth	80 sec
7.24c	100	Arc	0.1	0.1	1.2/0.1	Very Smooth	157 sec
7.24d	50	Arc	0.1	0.1	2.0/0.1	Rough	53 sec
7.24e	50	Arc	0.1	0.1	1.2/0.3	Moderate	80 sec

Table 7.2 : The effect of selected simulation parameters.

1. These numbers are ratios of the ideal segment length  $S_{id}$ . Two numbers are listed. The first is the ratio of the maximum size to the ideal size,  $S_{max}/S_{id}$  and the second is the ratio of the minimum allowable size to the ideal size  $S_{min}/S_{id}$ .
2. Computation time was measured on a SUN 4/280, for simulation of 30-second resist development with the etch-rate distribution of Figure 2.3. The data printed is the average of 3 data-points.
3. The profile in this simulation was incorrect, due to abrupt reflection of the etch-rays.
4. Figure 7.23c and 7.24b are identical.

step is used. In Figures 7.23a-b, there is a very noticeable change when the time-step is increased from 0.1 seconds to 0.5 seconds - the profile in the latter simulation is rougher and sharp-edged. However, the use of arc interpolation appears to offset this. As can be seen, the profiles of Figures 7.23c and 7.23d appear similar in shape and smoothness.

### Recursive Length

Recursive vector checking, discussed in Section 7.4.2, is very important for ensuring that the rays do not bounce abruptly when large time-steps are taken. This procedure uses recursive time-steps to calculate the trajectory of the rays with greater accuracy. The maximum change in the angle of the direction vectors of the rays is proportional to the recursive length. A recursive length of 0.1 corresponds to a maximum change of  $4^\circ$ , while a recursive length of 2 ensures that no recursive vector checking is used. Figures 7.23e and 7.23f demonstrate the effectiveness of recursive vector checking. If the time-step of the simulation is small to begin with (Figure 7.23e) recursive vector checking appears to have little effect on the accuracy of the simulation. But there is a slight increase in the computation time when recursive vector checking is used. However, when the time-step is large, the lack of recursive vector checking causes the rays to bounce and reflect off regions of high etch-rate slope.† As a result, the simulated profile, shown in Figure 7.23f, is incorrect. With recursive vector checking (Figures 7.23b,d), the profiles are relatively accurate even though the time-step is large. Therefore, it appears that recursive vector checking is well worth the price paid in increased computation time.

### Ideal Segment Length

---

† Recall from the discussion in Section 7.4.1 that the threshold time-step size for no abrupt ray reflection is 0.18 seconds. These simulations show that without recursive vector checking, the rays are accurate when the time-step is 0.1 seconds, but bounce for a time-step of 0.5 seconds.

The ideal segment length, or the number of segments on the initial mesh, does play an important role in the accuracy of the simulation. As might be expected, the accuracy increases as more segments (smaller segment lengths) are used initially in the simulation. This is easily seen in Figures 7.24a-c. But note in Table 7.2 that that the computation time increases linearly with the number of segments used.

#### Allowable Segment Length Range

The range in the allowable segment lengths also affect the accuracy of the simulation. This was previously discussed in Sections 7.5.1.1-2. To recap, if the maximum allowable segment length is too large, the simulation will be inaccurate, since there will be relatively sparse regions on the surface. The inaccuracy hurts particularly in photoresist development simulations, where the surface changes rapidly with distance. The profile in Figure 7.24d, where  $S_{\max} = 2.0S_{ideal}$  does look somewhat uneven. The minimum segment length also affects the accuracy of the profiles. If the minimum allowable segment is too long, then the segments will poorly approximate the surface. But as seen in Figure 7.24e, increasing the minimum allowable segment length from  $0.1S_{ideal}$  to  $0.3S_{ideal}$  does not appear to affect the simulated profile.

### 7.7. SUMMARY : 2D IMPLEMENTATION ISSUES

In this chapter, the issues affecting the accuracy and efficiency of the ray-string simulation have been examined in some detail. This was done with the aid of numerous 2D simulations using the ray-string algorithm.

Two important modifications to the ray-string algorithm have been developed based on the implicit constraints of the ray equation. The first is a rule-of-thumb procedure for select-

ing the threshold time-step for which abrupt ray reflection will not occur. In simulations involving periodic etch-rate distributions, this threshold time-step is one in which the accuracy and efficiency of the ray calculations are most optimized. The second procedure, recursive vector checking, is one that has a tremendous impact on the accuracy of the simulations. Recursive vector checking effectively increases the accuracy of large-time-step simulations by taking small recursive time-steps in which the change in the ray trajectories are limited to a few degrees. The price paid for the improvement in accuracy is that of computation time; the simulations reveal that the increase in computation time due to the recursive procedure is directly proportional to the size of the time step.

The ray-string algorithm also requires that a number of mesh operations be performed on the mesh during the course of the simulation. These operations are mesh modification, mesh boundary clipping, boundary vector flattening, and delooping. The comprehensive testing of the 2D ray-string algorithm has shown that the accuracy of the simulations is increased with the use of (a) small time-steps, (b) small segments lengths, (c) small maximum allowable segment length, (d) arc interpolation, and (e) recursive vector checking. In all cases except (d) arc interpolation, the increase in accuracy is accompanied by an increase in computational time.

It is said that knowledge is power. The rigorous testing of the ray-string algorithm has provided a great deal of information about the behavior of the ray-string algorithm and on the issues, procedures and operations that affect the accuracy and speed of the computations. This knowledge is of great value in the implementation of the ray-string algorithm in three dimensions.

## REFERENCES

1. A. Moniwa, T. Matsuzawa, T. Ito, H. Sunami, "A Three-Dimensional Photoresist Imaging Process Simulator for Strong Standing-Wave Effect Environment," *IEEE Transactions on Computer Aided Design*, vol. CAD-6, no. 3, pp. 431-437, May 1987. *TRIPS-I*: 3D Ray-String method.
2. E.W. Scheckler, June 1989. Personal communication, work in progress.
3. E. Barouch, B. Bradie, H. Fowler, S.V. Babu, "Three-Dimensional Modeling of Optical Lithography for Positive Photoresists," *Interface'89 : Proceedings of KTI Microelectronics Seminar*, pp. 123-136, November 1989. 3D Ray method.

## **CHAPTER 8**

### **THE RAY-STRING ALGORITHM : 3D IMPLEMENTATION ISSUES**

#### **8.1. INTRODUCTION**

Resist development algorithms using the cell, string, and ray approaches have been implemented and utilized to examine basic tradeoffs in accuracy, delooping requirements, and efficiency. A combined ray-string approach is by far the most advantageous, as it is fast, efficient and has modest memory requirements. The algorithm is also accurate and insensitive to errors in the local etch-surface. Consequently, the decision was made to implement the ray-string algorithm in three dimensions.

But the study of the algorithm in the previous chapter has revealed that the simulated mesh requires periodic mesh modification, boundary clipping and flattening, and an occasional deloop in order to preserve the balance between accuracy and efficiency. The information gleaned from the testing of the 2D algorithm proves to be of great aid in implementing the mesh operations in 3D. As will be shown in this chapter, the mesh implementation issues explored in 2D do carry over to 3D as well. But the 3D implementation is considerably more difficult, as there are a number of 3D "traps" which must be avoided. In addition, a careful choice of the data structure must be made so as to enable the accurate and robust implementation of these essential mesh operations.

## **8.2. IMPLEMENTATION OF THE RAY-STRING ALGORITHM IN 3D**

### **8.2.1. Data-Structure Requirements**

The first step in the implementation of the ray-string algorithm in 3D is to select a data-structure that can be used to efficiently represent a time-evolving surface. In the ray-string algorithm, the most important objects on this surface are the nodes or points, since it is these nodes that make up the etching surface. The data-structure must reflect this fact. Therefore, the data-structure must, at the very least, consist of a list of nodes. Each node in turn must contain the coordinates and direction vector of the node.

But other than the basic node information, what other data must the data-structure hold? The answer to this question lies in the various operations that have to be carried out on the surface mesh. These operations are listed below, in order of importance.

#### **Mesh Modification**

In the interests of efficiency and accuracy, the density of nodes on the mesh has to be controlled. The mesh has to be modified so that additional nodes are added to node-scarce regions, while nodes that are too close together are deleted.

#### **Mesh Clipping**

If the mesh expands outside the simulation boundaries, then it should be clipped. This is really necessary to control the size of the mesh. Without clipping, a mesh could grow outside the simulation boundaries. Nodes outside the simulation boundaries do not contribute to the simulation in the area of interest, so it would be inefficient to continue tracking these nodes.

#### **Mesh Delooping**



In both the ray and string algorithms, it is quite possible for the surface to form loops. The same is true of the ray-string algorithm. The loops formed are non-physical in nature, and should be deleted for correctness.

### **Mesh Plotting**

The graphical interface is also of some importance, for it is an invaluable tool for interpreting the simulated data. The data-structure must be set up so that data can easily be plotted.

Given that all the above mesh operations are necessary to retain the accuracy and correctness of the mesh, what data structure can best be used to implement these operations accurately and efficiently?

### **8.2.2. 3D Data-Structures**

After some consideration, it was decided that the essential mesh operations previously described could best be implemented using a hierarchical structure of nodes, segments and triangles. Each object in this data-structure has its uses. The nodes contain the surface coordinate and vector information. The segments are used for mesh modification and clipping. And the triangles are used in the 3D mesh delooping and plotting operations.

The 3D ray-string algorithm has been implemented in the C programming language, using a linked list data structure to represent the nodes, segments, and triangles that make up the etching boundary. This data-structure is shown in Figure 8.1. As shown, each triangle contains 3 segments, and each segment in turn contains 2 nodes. Each node-segment-triangle interconnection is two-way. A triangle has pointers to the three segments attached to it, and each segment has pointers to its neighboring triangles. The same applies to the nodes; each

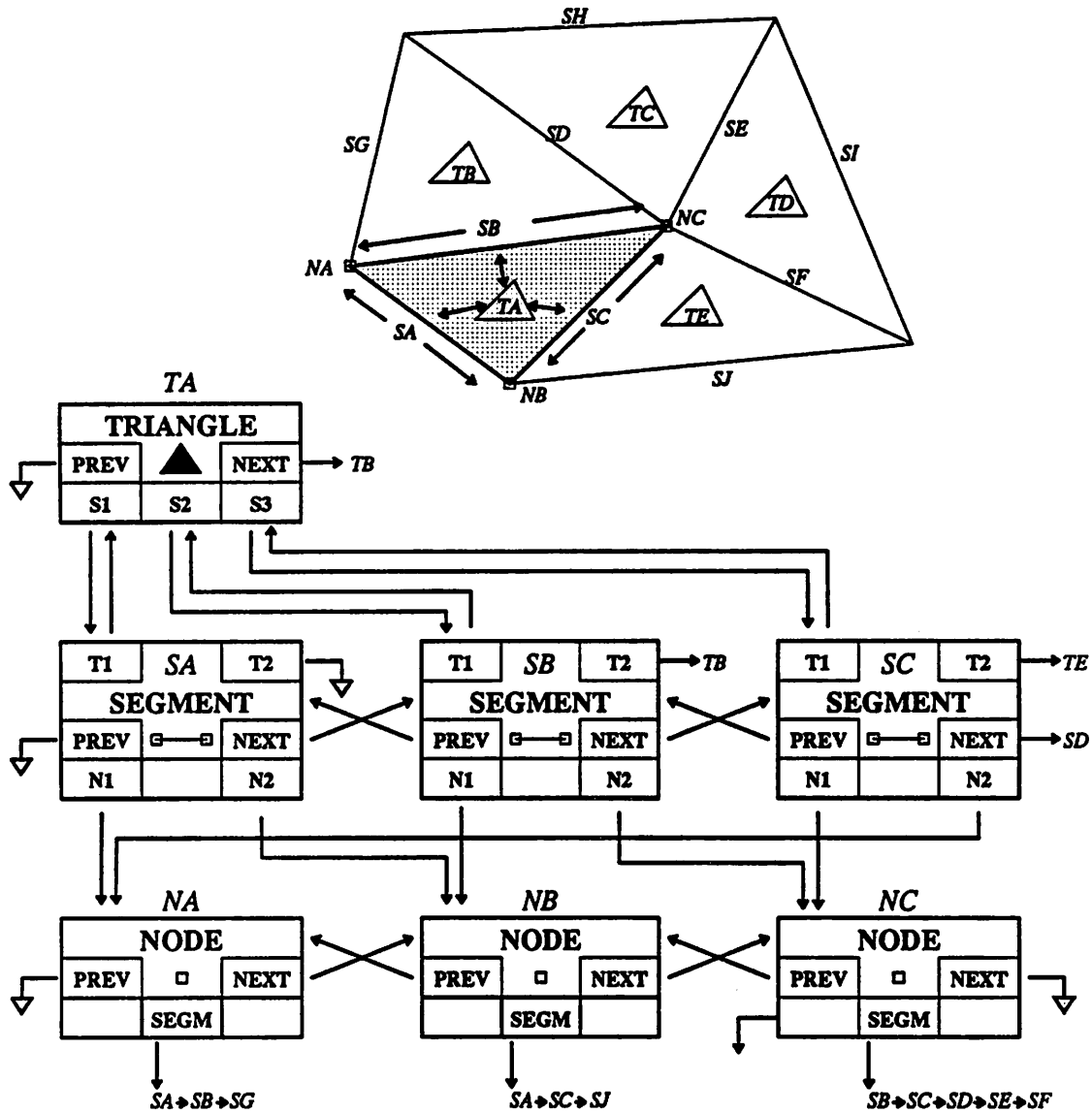


Figure 8.1 : Data structure used for 3D ray-string algorithm. The data structure consists of linked lists of nodes, segments, and triangles. Each triangle is connected to 3 segments, while each segment is connected to 2 nodes. Each segment also has a link to its neighboring triangles, while each node maintains a list of its neighboring segments.

node contains an array of pointers to its neighboring segments. This last interconnection is necessary to retain mesh integrity during mesh modification, clipping and delooping.

### **8.3. THE ALGORITHM FOR SURFACE ADVANCEMENT**

The procedure for advancing the surface mesh in 3D is identical to that described for 2D in the previous chapter. To recap, the simulation begins by creating and initializing a surface consisting of a number of interconnected nodes, each with a ray vector normal to the initial surface. The algorithm then proceeds as follows :

- [I] Advance each node on the boundary mesh along its direction vector, and calculate the new direction vector at the new node location using the discretized form of the differential ray equation. Use recursive vector checking for more accurate calculation of the node trajectory. In addition, since boundary nodes must not be allowed to enter the simulation volume, the boundary vector nodes must be flattened if necessary.
- [II] Add nodes in regions where the mesh has expanded, and delete nodes in regions of contraction.
- [III] Clip the mesh if it extends outside the boundary.
- [IV] Deloop the mesh if necessary. An algorithm for delooping the mesh will be described and demonstrated in a later section.
- [V] Proceed to [I] and repeat until the total etch-time has been reached.

## **8.4. MESH OPERATIONS**

Mesh operations are a necessary but distasteful ingredient of the ray-string algorithm. These operations must be carried out to ensure that the mesh remains accurate and efficient and manageable. But these operations are also not easy to implement in 3D, primarily because of the complications involved in manipulating a three-tiered data-structure such as is used in the ray-string approach.

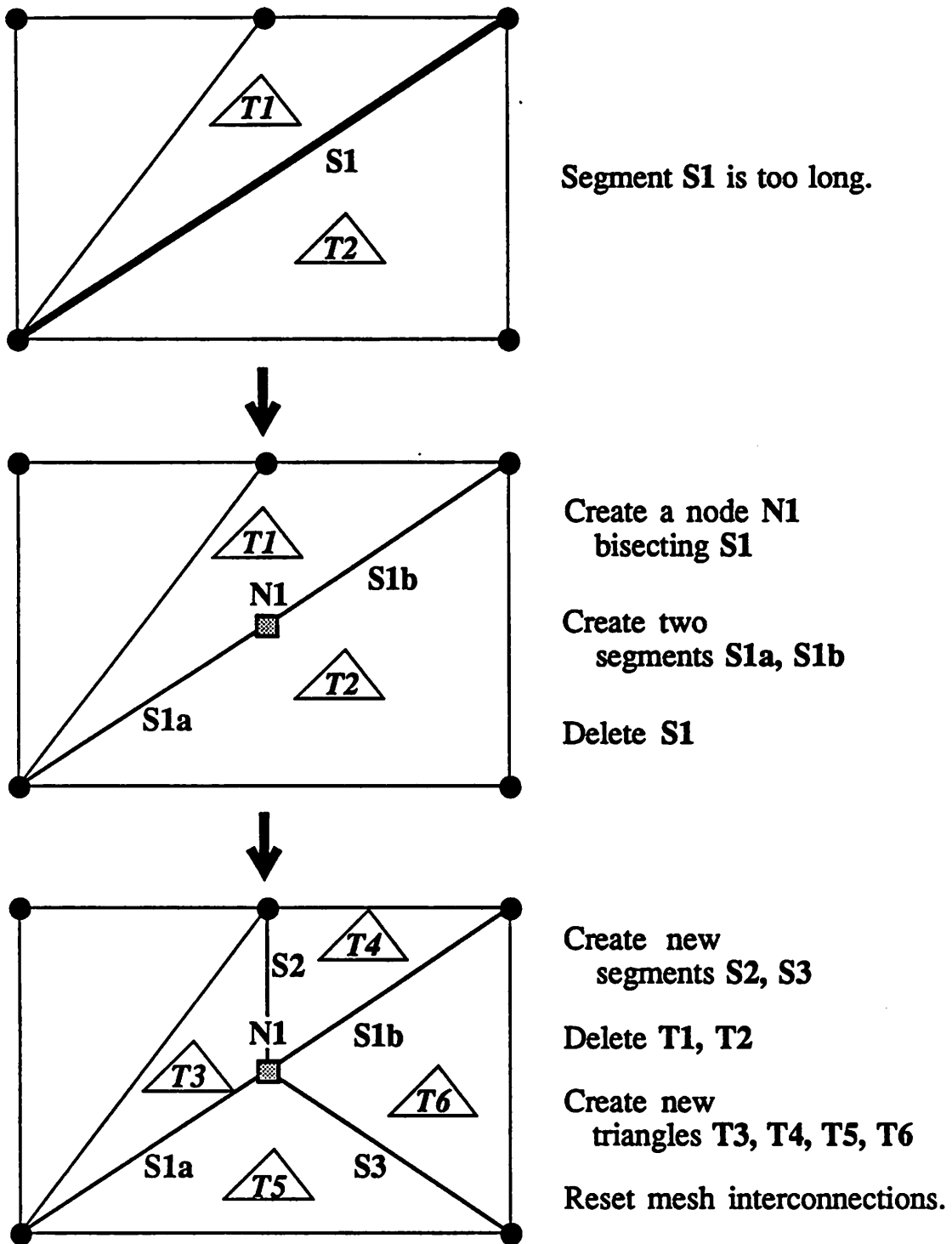
### **8.4.1. Mesh Modification**

Mesh modification is as important in 3D as in 2D. During the course of the simulation, the nodes on the surface mesh will move farther apart or closer together. In order to retain a balance between accuracy and efficiency, nodes have to be added to or deleted from the mesh.

As in 2D, mesh modification in 3D is segment-based. The connection between the nodes on the mesh is maintained by a list of segments, so, in order to control the density of nodes in the mesh, segments have to be cut up if they become too long or deleted when they become too short. This procedure is somewhat complicated in 3D, because after mesh-modification, the interconnections between the triangles, segments and nodes in the mesh must be updated.

#### **8.4.1.1. Segment Cutting**

Segment cutting is somewhat more difficult in 3D than in 2D. Figure 8.2 depicts the steps that are taken whenever an over-long segment is discovered. First, a new node bisecting the segment has to be created. This node could be placed using linear interpolation, or as in



**Figure 8.2:** 3D Segment Cutting. If a segment is too long, it must be cut up into smaller segments. This entails creating new nodes, segments and triangles. The interconnections between the nodes, segments and triangles must then be reset.

the 2D case, it could be placed on an arc defined by the direction vectors of the two nodes on the segment. The segment is then cut up and replaced by two new segments. New triangles are also created during this process. And finally, interconnections between the nodes, segments and triangles are updated to reflect the changes made in the mesh.

Figure 8.3 shows 3D profiles from simulations of a uniform spherical etch. The plots show the mesh profile after 10 0.1-second time-steps; the solid dark lines superimposed on the top-view plots are the expected circular result. The top pair of plots use linear interpolation during the segment cutting procedure, while the lower pair use arc interpolation for node addition. It is not difficult to see that the arc interpolation produces "fuller" and more accurate results. In the top-view plots of Figure 8.3, it is readily seen that the linearly interpolated mesh falls short of the expected circular result. In contrast, when arc interpolation is used to modify the mesh, the resultant mesh is more accurate and quite satisfactorily spherical.

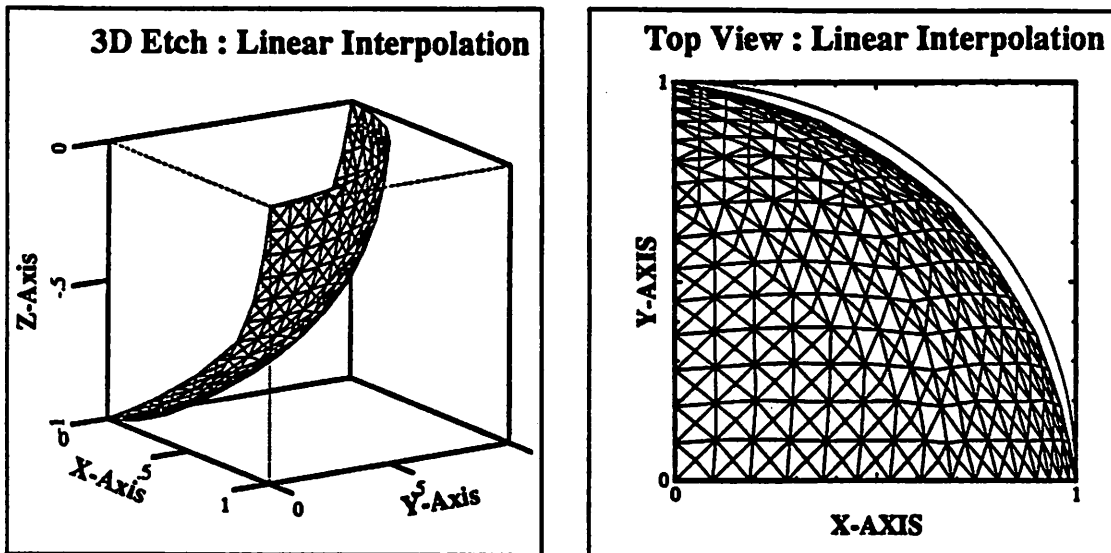
#### 8.4.1.2. Segment Merging

The procedure for 3D node deletion, shown in Figure 8.4, is based upon merging short segments. This means that if a segment is deemed too short, i.e.,  $S < S_{\min}$ , then one of the nodes on that segment is deleted.† Segments and triangles common to the deleted node and the short segment are also deleted. At the same time, the neighbors of the deleted node are reconnected to the undeleted node on the short segment. The original short segment is then deleted. The final step is to reset the interconnections between the nodes, segments and triangles on the mesh.

---

† The algorithmic decision on the node to be deleted is made based on the number of segments attached to the node - the deleted node is that with the most segments attached. This implicitly controls the number of triangles attached to a single node. In addition, the node to be deleted must *not* be on the boundary.

## LINEAR INTERPOLATION



## ARC INTERPOLATION

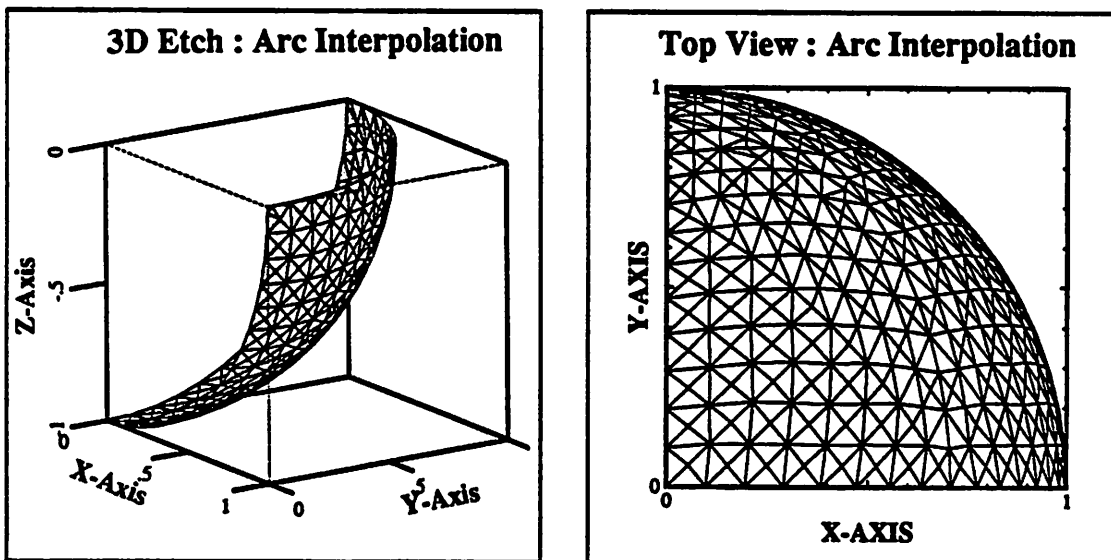
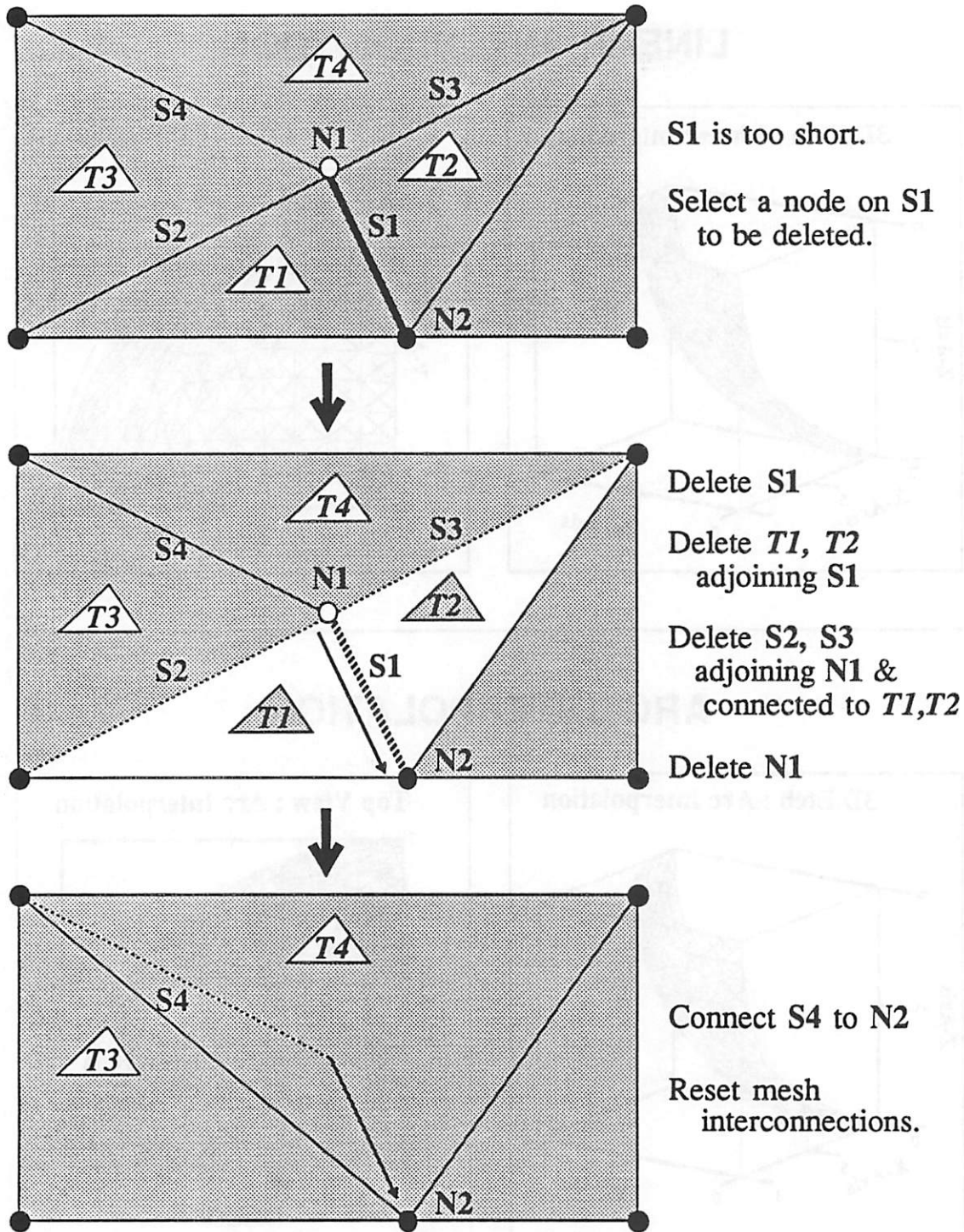


Figure 8.3 : Simulation of uniform spherical etching with the ray-string algorithm, using linear interpolation and arc interpolation. The etch begins from a single seed point at coordinates (0,0,0), and proceeds for 1 second of etching time. The time-step is 0.1 sec.



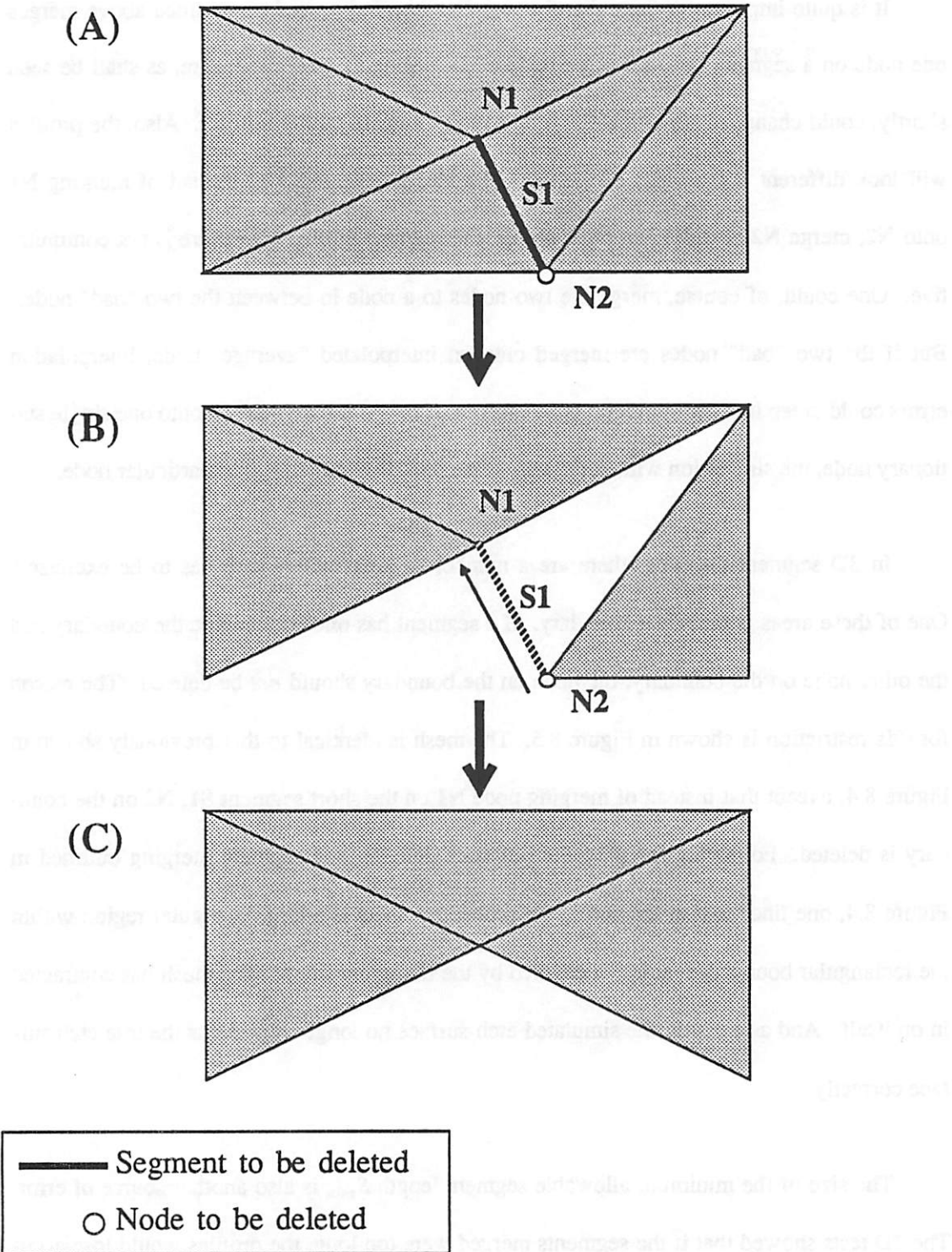
**Figure 8.4 :** 3D Segment Merging. A segment is merged by first selecting a node on the segment to be deleted, then deleting the segments and triangles common to the node and to the triangles adjoining the segment. The interconnections between the nodes, segments and triangles then must be reset.



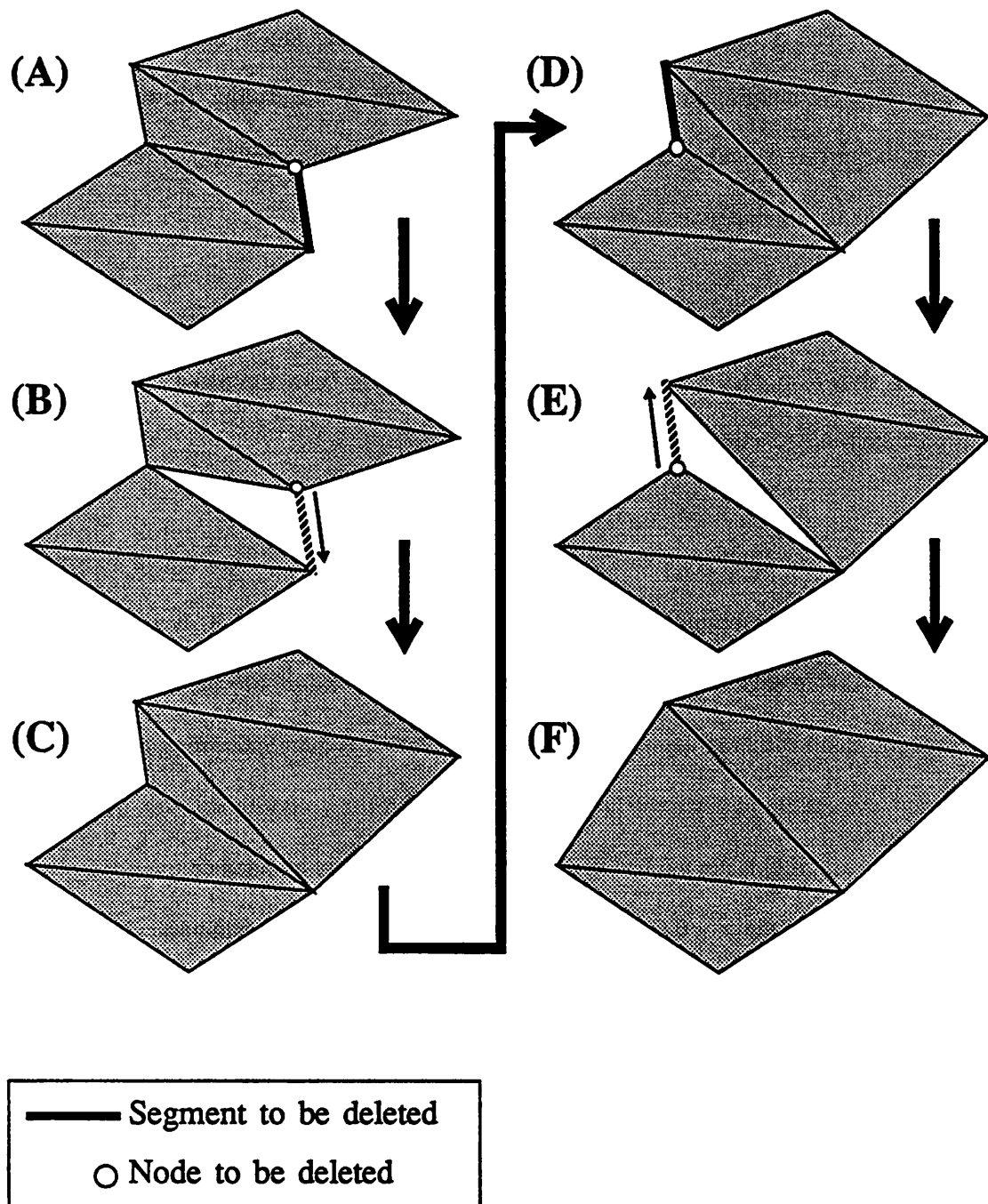
It is quite important to note that the segment-merging procedure outlined above merges one node on a segment onto the other node of the segment. This procedure, as shall be seen shortly, could change symmetrical 3D profiles to asymmetrical 3D profiles. Also, the profiles will look different if the order of the merged nodes is changed, i.e., instead of merging N1 onto N2, merge N2 onto N1. In other words, the segment cutting procedure is not commutative. One could, of course, merge the two nodes to a node in between the two "bad" nodes. But if the two "bad" nodes are merged onto an interpolated "average" node, interpolation errors could creep into the simulation. In contrast, if the nodes are merged onto one single stationary node, the simulation will, at the very least, remain accurate at that particular node.

In 3D segment meshing, there are a number of areas where care has to be exercised. One of these areas is the mesh boundary. If a segment has one node inside the boundary and the other node on the boundary, the node on the boundary should *not* be deleted. The reason for this restriction is shown in Figure 8.5. The mesh is identical to that previously shown in Figure 8.4, except that instead of merging node N1 on the short segment S1, N2 on the boundary is deleted. Following the procedure of node-deletion and segment merging outlined in Figure 8.4, one finds that at the end of the procedure, there is a large triangular region within the rectangular boundary that is not covered by the triangular mesh! The mesh has contracted in on itself. And as a result, the simulated etch-surface no longer represents the true etch surface correctly.

The size of the minimum allowable segment length  $S_{\min}$  is also another source of error. The 2D tests showed that if the segments merged were too long, the profiles would lose accuracy. The same is true in 3D. An example is shown in Figure 8.6. The mesh initially consists of six triangles on a sharp stepped surface. There are two short segments on opposite vertical sides of the step. Now, suppose that one segment is merged upwards (i.e. the lower node is



**Figure 8.5 :** 3D Segment merging. If the node deleted is on the boundary of the mesh, the mesh will contract in on itself. This is undesirable.



**Figure 8.6 :** 3D Segment merging on a folded surface. If the short segments marked in thick dark lines are merged, the resultant surface will be asymmetrical and inaccurate.

merged onto the upper node) while the other short segment is merged downwards. Then, as shown in Figure 8.6, after the segment merging procedure, the resultant mesh will become asymmetrical! The profiles of Figure 8.6a and 8.6f do not bear much resemblance to each other; the segment merging procedure has in effect "averaged" the profile. However, if only very small segments are merged, the averaging error will have a lesser impact on the mesh. Therefore, it is best to merge or delete only very small segments.

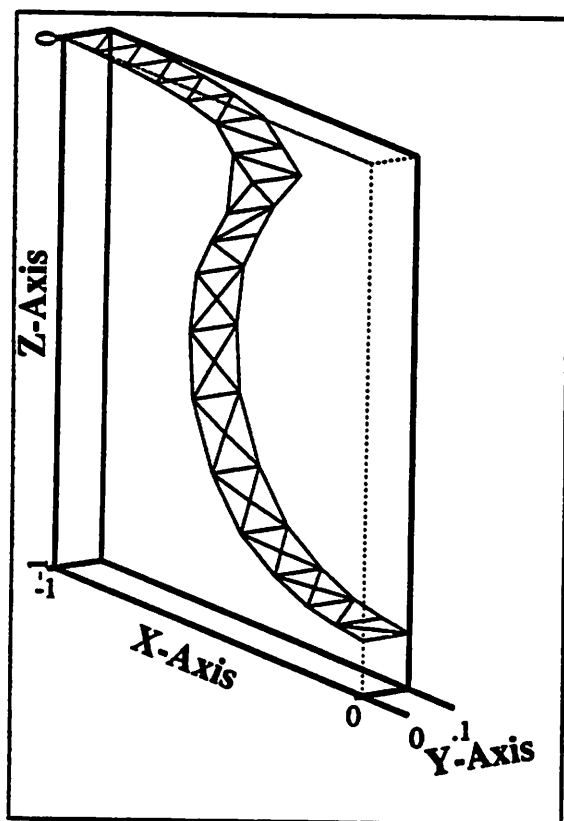
A real example of the symmetrical-to-asymmetrical profile conversion with segment deletion is shown in Figure 8.7, where the 3D profile has been simulated using the 2D sinusoidal etch-rate function

$$R(x,y,z) = e^{-4x^2}(1.05 - \cos(2\pi z)) \quad [8.1]$$

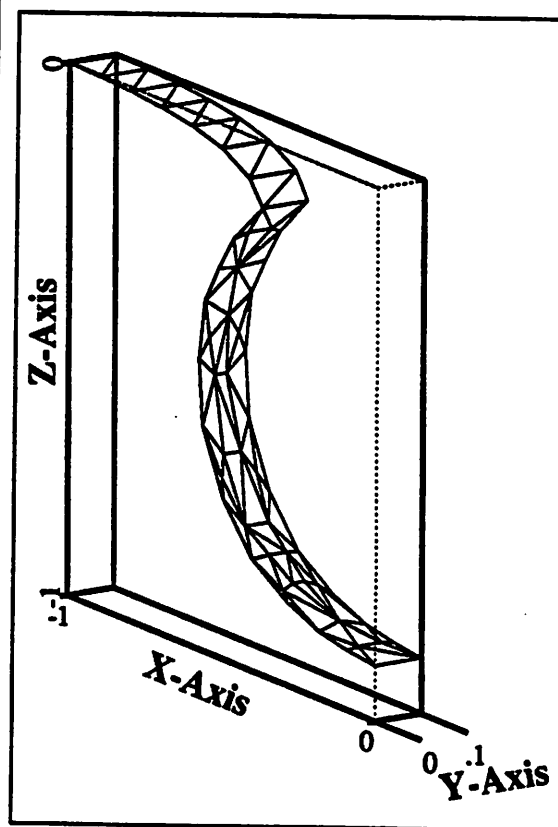
The simulations were both run for 200 time-steps of 0.01 seconds. Figure 8.7a shows the profile with a minimum segment length of  $S_{\min} = 0.7S_{ideal}$ , while Figure 8.7b uses a smaller minimum allowable segment length of  $S_{\min} = 0.2S_{ideal}$ . The feature of interest in these two figures is the sharp corner at the upper portion of the plot. It is quite interesting to see that in this region, the profile in Figure 8.7a is not constant with the  $y$ -coordinate even though the etch-rate function [8.1] is not a function of  $y$ . The profile appears rounded on the  $y = 0$  plane, and sharp-edged on the  $y = 0.1$  plane. This profile is a result of asymmetrical segment-merging; a segment on the  $y = 0$  plane has been merged in a different direction than its corresponding segment on the  $y = 0.1$  plane. A much better profile is produced when a shorter  $S_{\min}$  is used, as shown in Figure 8.7b. However, note that in this case, there are several obtuse triangles at the bottom portion of the mesh. These triangles are formed when long segments are cut; since only very short segments are merged, the triangles remain obtuse.

It is often said that a "good" mesh is one that consists primarily of equilateral triangles. This may be true for finite-element analyses, but not for mesh-based surface-advancement

(a)  $S_{\min} = 0.7 S_{idl}$



(b)  $S_{\min} = 0.2 S_{idl}$



**Figure 8.7:** 3D ray-string etching simulation using a sinusoidal etch-rate function. The simulation proceeds for 200 time-steps of 0.01 second duration each. The simulations in (a) and (b) differ in the size of the shortest allowable segment length. Note that the profile in (a) has a y-dependence even though the etch-rate function is independent of y.

algorithms. Figures 8.6 and 8.7 indicate that it is not realistic to expect accurate results with equilateral triangles. To obtain equilateral triangles, or approximately equidistant nodes, the segment length ratios  $S_{\min}/S_{ideal}$  and  $S_{\max}/S_{ideal}$  must be close to 1. But as seen in Figure 8.7a, if the segment length ratio  $S_{\min}/S_{ideal}$  is relatively large, the simulation produces an inaccurate profile. This reflects the fact that equilateral triangles are not suitable for approximating rough irregular surfaces. Such surfaces are better described by triangles with unequal sides. A segment length ratio  $S_{\min}/S_{ideal} = 0.2$  produces uneven triangles which don't look very eye-pleasing, but at least the mesh does represent the surface accurately!

It is also worth mentioning that the short  $S_{\min}$  simulation (Figure 8.7b) used up approximately 7.1 CPU seconds on a SUN 4/280, whereas the long  $S_{\min}$  simulation (Figure 8.7a) cost 6.4 seconds. The use of smaller allowable segments increased the accuracy of the simulations, but at a cost of an increase of about 10% in the computation time. This is one aspect of the efficiency vs. accuracy tradeoff.

Another problem specific to 3D surface remeshing is that of zero-volume surfaces, as shown in Figure 8.8. The situation is as follows. Suppose that during the evolution of the mesh, a triangular pyramid composed of three smaller triangles has been formed, as shown in Figure 8.8a. Now, if a segment at the base of this pyramid is merged or deleted, the segment merging procedure will result in a folded surface, in which two triangles share the same nodes and segments. In other words, the pyramid has been flattened into a double-triangle! The flattened pyramid is depicted in Figure 8.8c. This result is not only physically incorrect, but it could also cause computational problems further on during the simulation. To avoid this particular problem, it is necessary to check each segment before it is merged, to find out if the segment to be merged is part of a 3-triangle pyramid. If the test is positive, the segment should not be merged.

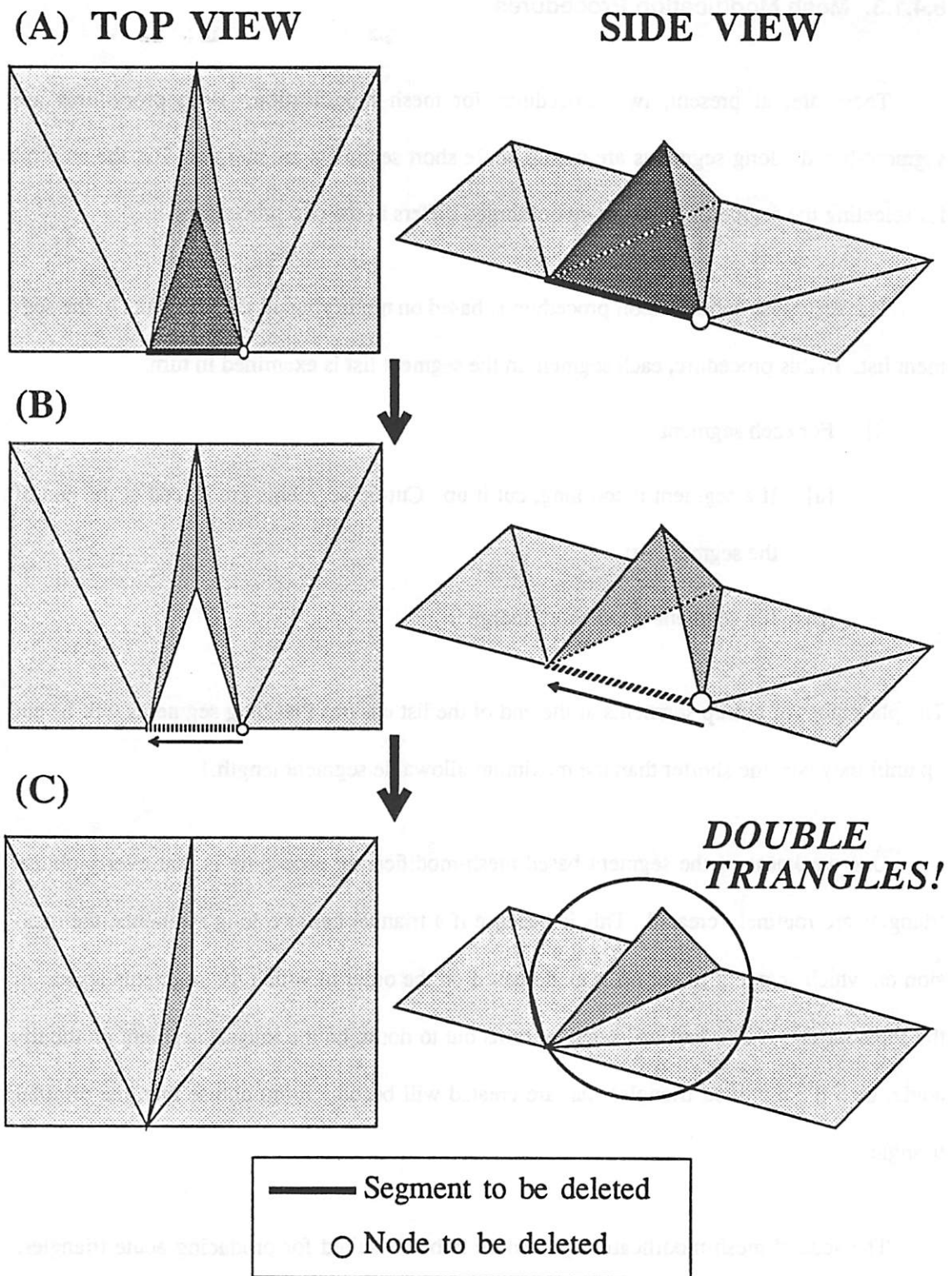


Figure 8.8 : 3D Segment merging. If a segment to be deleted is part of a 3-triangle pyramid, merging that segment will result in a flat fold where 2 triangles share the same segments and nodes.

### 8.4.1.3. Mesh Modification Procedures

There are, at present, two procedures for mesh modification. Both procedures are segment-based - long segments are cut up while short segments are merged. But the method for selecting the segments to be cut up or merged differs in the two procedures.

The first mesh-modification procedure is based on modifying all the segments in the segment list. In this procedure, each segment in the segment list is examined in turn.

- [I] For each segment :
  - [a] If a segment is too long, cut it up. Cut-up segments are placed at the end of the segment list.
  - [b] If a segment is too short, merge it.

The placement of cut-up segments at the end of the list ensures that long segments will be cut up until they become shorter than the maximum allowable segment length.†

One weakness of the segment-based mesh-modification procedure is that overly-obtuse triangles are routinely created. This is because if a triangle has two long segments, the decision on which segment to cut up is made based on the order in which the segments appear in the segment list. If this "cut-up" segment turns out to not be the longest segment on the triangle, then the modified triangles that are created will become more obtuse than the original triangle.

The second mesh-modification procedure is better suited for producing acute triangles.

---

† Note that short segments created from cut-up long segments are not merged unless these segments occur later on in the segment list. This avoids an infinite loop.



This procedure is based on examining the segments on a triangle.

- [I] Each triangle in the triangle list is examined in turn. For each triangle :
  - (a) If the longest segment on the triangle is too long, then the segment is cut up.
  - (b) If the above condition is not met, then the segments are checked for shortness. If the shortest segment on the triangle is too short, then it is merged.
  - (c) If neither of the above are true, then the segments are in the correct range and the triangle is left unmodified.

This procedure results in reasonably well-shaped triangles. It does not guarantee the production of only acute triangles, but at least the triangles that have been modified do not become over-obtuse in shape.

#### 8.4.2. Mesh Clipping

The mesh should be clipped periodically to trim off those parts of the mesh that have moved outside the simulation boundaries. This is not too difficult to do in two dimensions. However, in 3D, clipping can be a very expensive and difficult operation. As with mesh modification, the mesh can be clipped by cutting up segments. Figure 8.9 shows the basic procedure for 3D clipping. First, all the segments are tested for intersections with a boundary. Segments that do intersect the boundary are cut up, and new segments and triangles are created. This procedure is repeated for the six borders of the 3D box. Finally, the triangles, segments and nodes are checked; any of these objects outside the 3D boundaries are deleted. An example of clipping is shown in Figure 8.10, where a mesh containing 100 triangles is clipped. Note that the clipping procedure itself creates many small triangles. These triangles can be removed, if desired, by following the clipping operation with a mesh modification step.

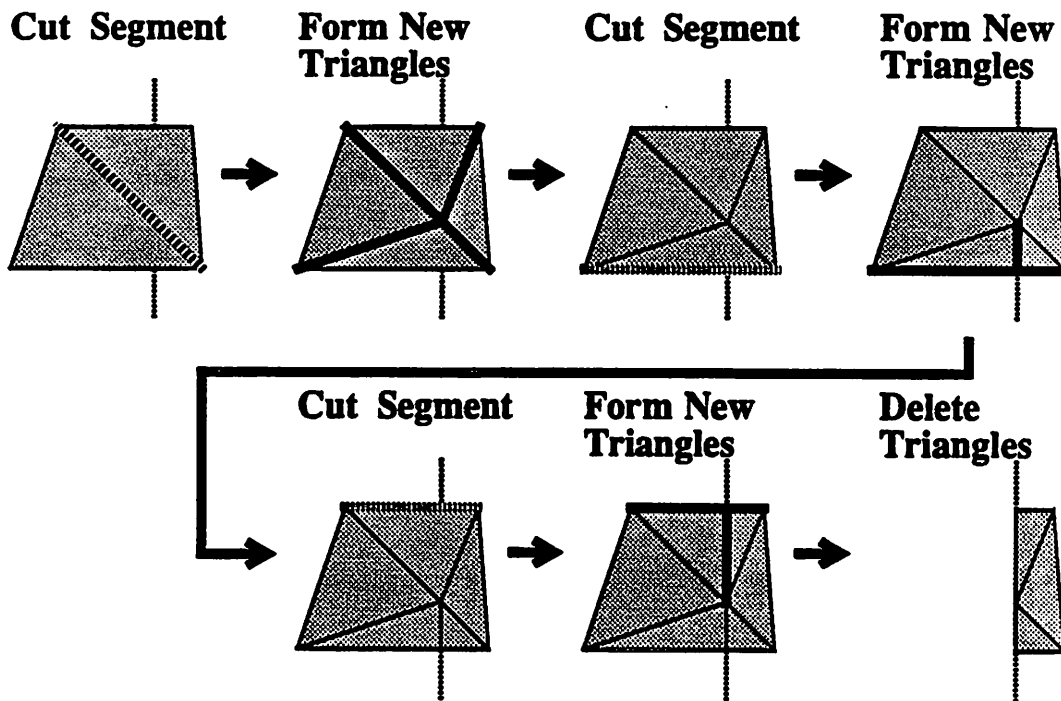


Figure 8.9 : 3D Clipping : All the segments that cross a boundary are cut up in turn. Each segment cut produces new triangles. Finally, the triangles, segments and nodes outside the boundary are deleted.

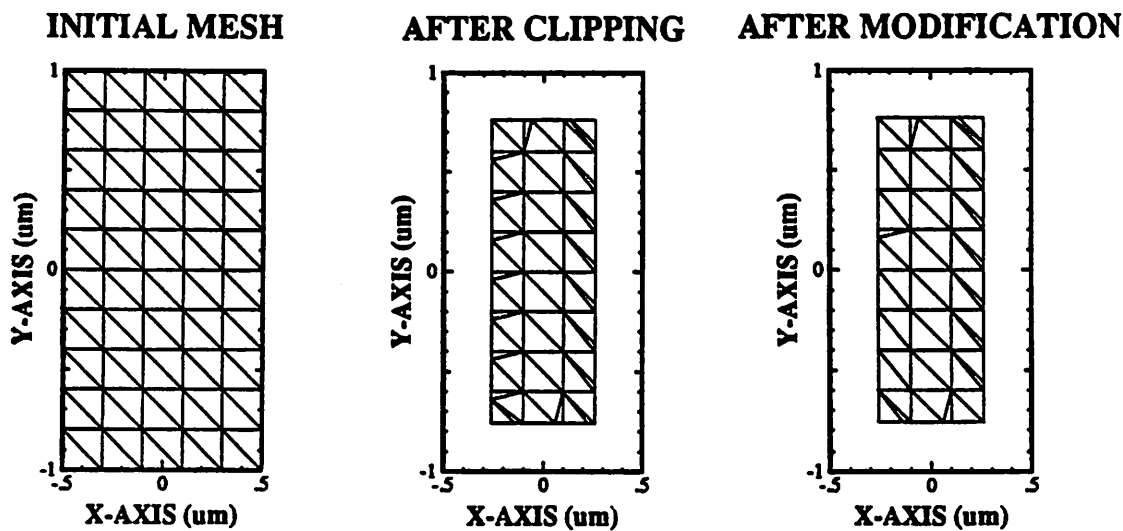


Figure 8.10 : 3D Clipping : An initial mesh containing 100 triangles is clipped from  $x=-0.26$  to  $x=0.26$ , and from  $y=-0.76$  to  $y=0.76$ . The obtuse triangles can be removed by mesh modification. The minimum allowable segment length is 0.04  $\mu\text{m}$ .

Clipping is expensive in 3D primarily because its basic component, segment cutting, is itself a lengthy procedure. When a single segment is cut up, triangles and segments have to be created and deleted, and interconnections to the mesh have to be updated. A typical mesh with  $50 \times 50$  nodes would contain perhaps 7500 segments; each of these segments has to be checked for intersections with the six boundaries. Clipping this mesh would be quite time-consuming. But at the same time, if the majority of these nodes have moved outside the simulation boundaries, it is wasteful to continue tracking their movement. Therefore, again, there is a tradeoff. Clipping removes useless nodes that are outside the simulation boundaries, thus saving computation time and memory. But at the same time, the clipping operation itself is time-consuming. So, there is a tradeoff between the time saved by deleting useless nodes and the time consumed during clipping. The compromise is, as previously mentioned, to clip only periodically, after a selected number of iterations.†

#### 8.4.3. Mesh Delooping

The procedure for removing loops in 3D follows very much along the lines of 2D delooping. But as might be expected, 3D delooping is a great deal more difficult, because it is now necessary to find the intersection of triangles in three-dimensional space. The very first step in 3D delooping is to examine all the triangles in turn and determine if these triangles do intersect other triangles. Intersecting pairs of triangles are cut up as outlined in Figure 8.11. The line of intersection of the two triangles is first determined. The triangles are then cut up at that intersection, by cutting up the segments of the triangle that cross the line of intersection. New triangles and segments are formed. The next step is to find the nodes at the

---

† At present, the mesh is clipped every 10 time-steps. A more intelligent approach would be to vary the clipping frequency with the local etch-rate. An alternative approach would be to find the bounds of all the nodes before clipping; if the outermost node is reasonably close to the boundary, the clipping could be delayed.

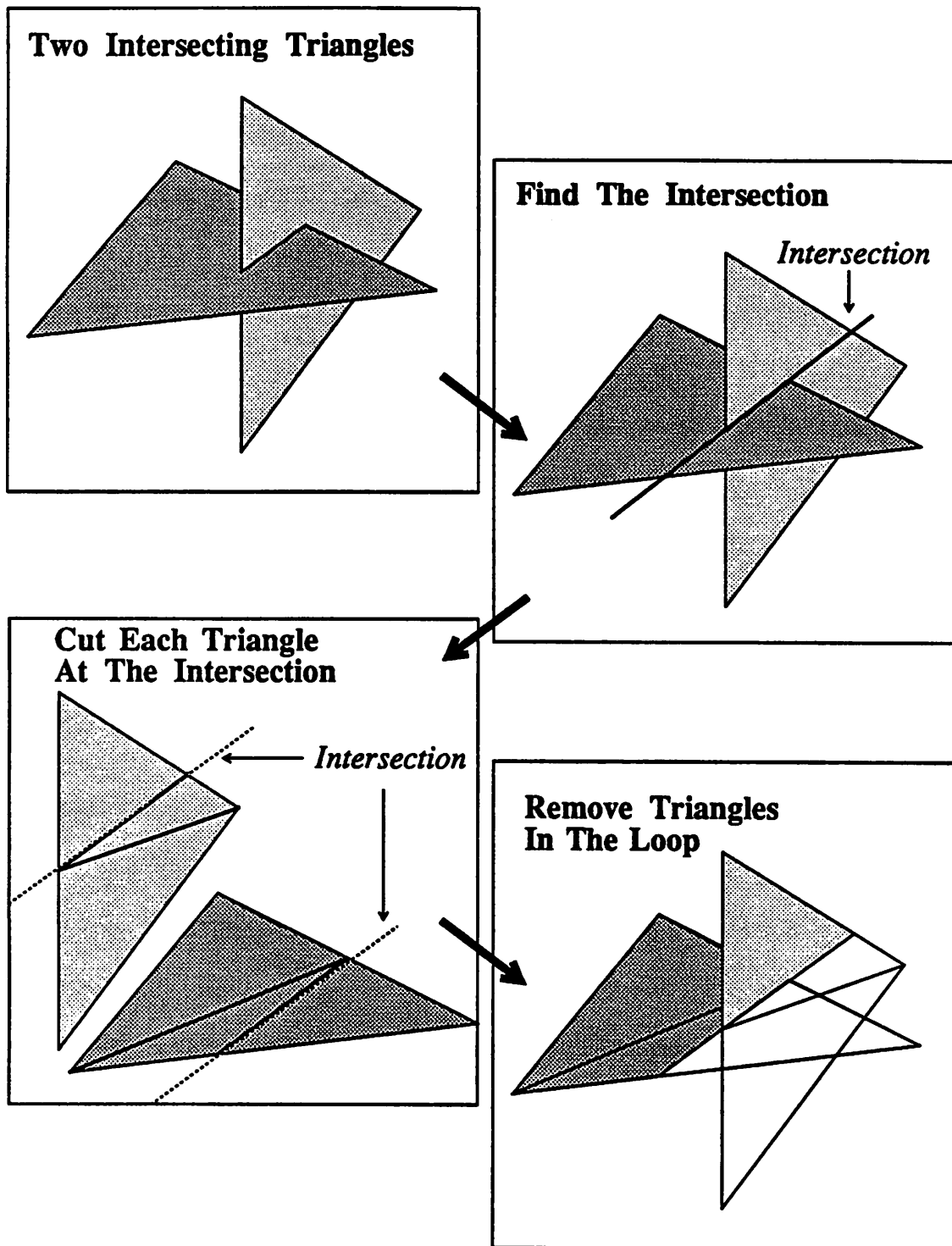


Figure 8.11 : 3D Delooping. The triangles are cut up at their intersections. The triangles in the loop are then removed.

boundary of the loop. This can be done as shown previously in Figure 7.18 by examining the motion of the intersecting triangles relative to their nodes. Once these boundary nodes have been specified, the loop is then traversed recursively to find and identify the triangles, segments and nodes in the loop. Finally, all the structures in the loop are removed.

The deloop procedure outlined above is illustrated in the etching simulation shown in Figure 8.12. In this example, the 3D ray-string algorithm was used to move the etch-front for 8 time-steps with the triangular etch-rate function

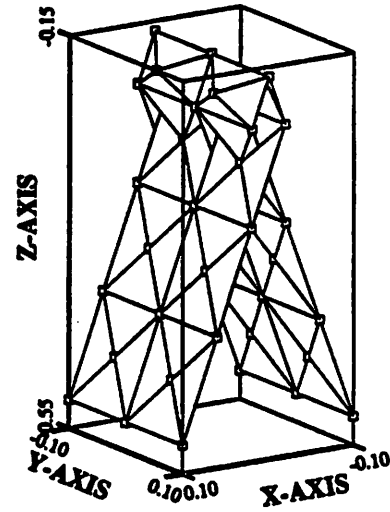
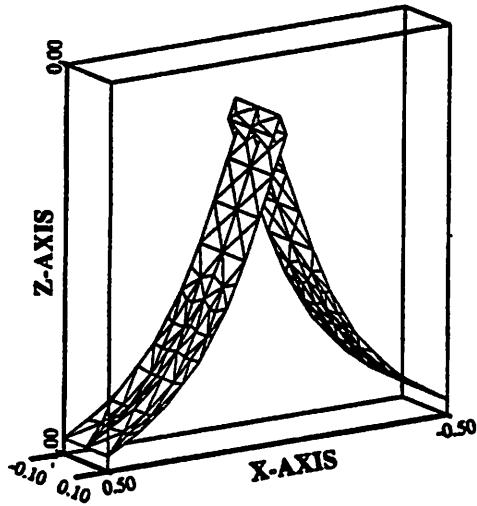
$$R(x,y,z) = 2|x| + 0.2 \mu\text{m}/\text{sec}, \quad |x| < 0.5 \quad [8.2]$$

The mesh was then delooped. The intersecting triangles were first cut up so that the triangles intersected cleanly at the boundary of the loop. The loop triangles, segments and nodes were then identified and deleted.

Another example of delooping is shown in Figure 8.13. In this case, the simulation is of two intersecting spherical etch-fronts, beginning from seed-points at (0,0,0) and (0,1,0). The etch proceeds at a uniform 1  $\mu\text{m}/\text{sec}$ , and each time-step is of 0.1 second duration. The spheres first intersect each other after 5 time-steps. At 8 time-steps, the mesh was delooped; the sections of the mesh behind the two spheres were removed. The mesh was then advanced again. At the final etch-time, two time-steps later, the mesh was found to have intersected again. The mesh was then delooped, resulting in the curved profile in the bottom left of Figure 8.13.

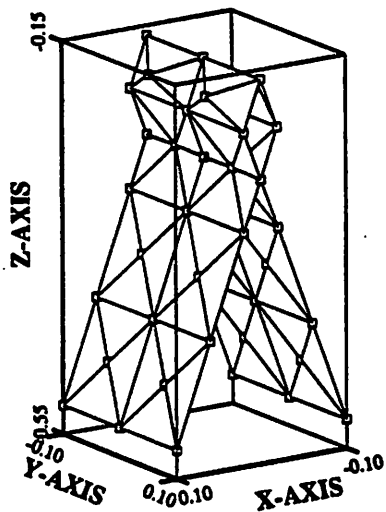
Approximately 20 CPU seconds on a SUN 4/280 were required to obtain the delooped profile on the bottom left of Figure 8.13. In contrast, a similar simulation using the modified cell algorithm (Figure 4.3b) lasted some 4 hours on the same computer. The profiles from the two different etching simulations are quite similar, except for the facets on the spherical

Triangular Rate Function :  $R(x,y,z) = 2|x| + 0.2$   
 8 TIME STEPS, BEFORE DELOOP      8 TIME STEPS, BEFORE DELOOP



CLOSEUP

CREATE NEW TRIANGLES



REMOVE TRIANGLES IN LOOP

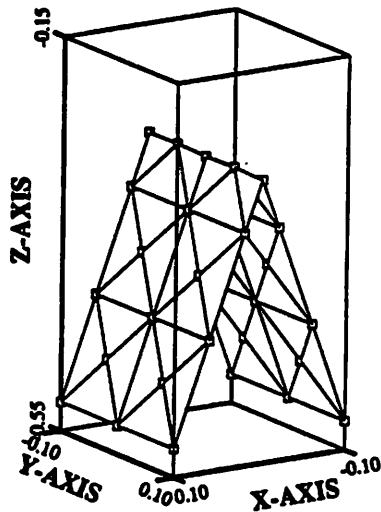
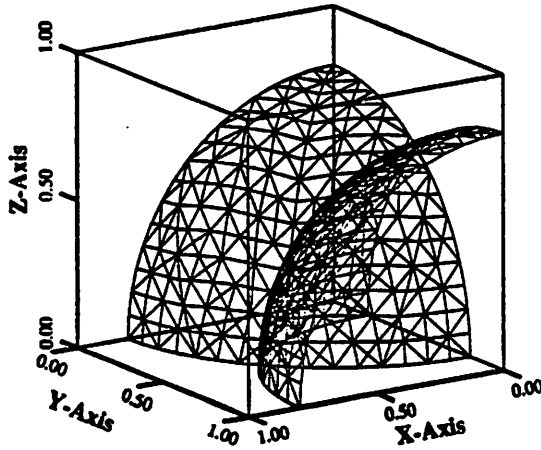


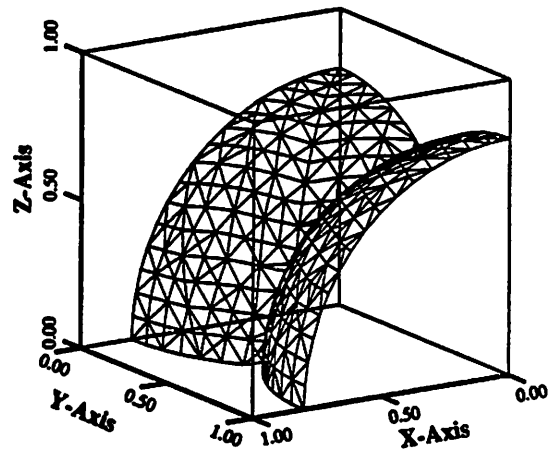
Figure 8.12: 3D ray-string etching using the triangular etch-rate function  $R=2|x|+0.2$ . The deloop procedure after 8 time-steps is illustrated.

Uniform Etch Rate : 2 Seed Points at (0,0,0) & (0,1,0)

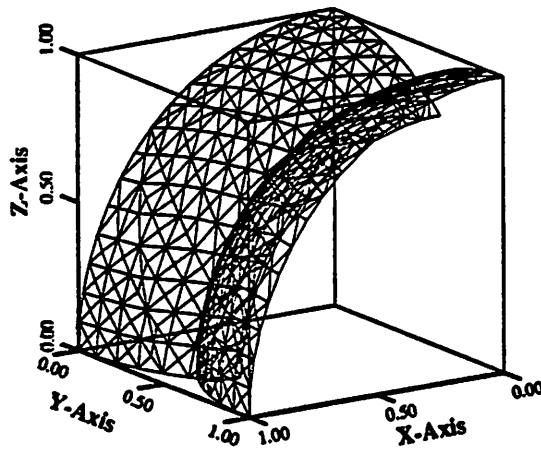
8 TIME STEPS, BEFORE DELOOP



8 TIME STEPS, AFTER DELOOP



10 TIME STEPS, BEFORE DELOOP



10 TIME STEPS, AFTER DELOOP

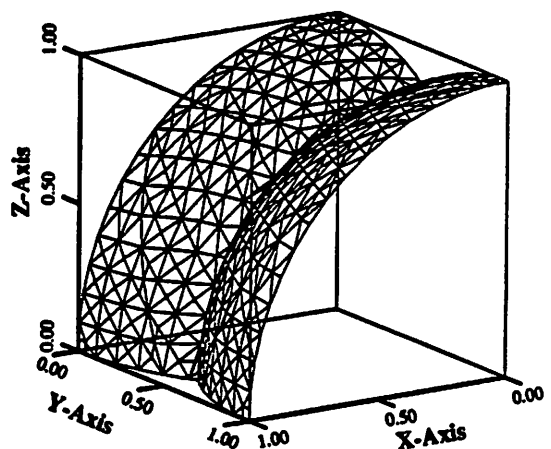


Figure 8.13 Uniform spherical etch using the ray-string algorithm. The mesh is delooped after 8 and 10 time-steps. The etch-rate is 1  $\mu\text{m}/\text{sec}$  and the time-step is 0.1 seconds long.

surfaces of the modified-cell profile. But unlike the ray-string algorithm, the modified cell profile did not have to be delooped, since the volume algorithm automatically takes care of intersecting etch-fronts. However, the modified-cell algorithm is slow. The ray-string algorithm, even with delooping, enjoys a tremendous speed advantage over the modified cell algorithm. This advantage alone makes the ray-string algorithm very attractive for 3D simulations.

### 8.5. TO DELOOP OR NOT TO DELOOP

It cannot be denied that 3D delooping in the ray-string algorithm is a problem of considerable computational complexity. Delooping also consumes a great amount of computation time. The ray-string simulation of the intersecting spheres described in the previous section went relatively fast because it involved only approximately 200 triangles. But in a typical simulation of photoresist development, a mesh could contain several tens of thousands of triangles. It is no easy task to determine the intersection of so many triangles. † Neither is it easy to do so robustly.

The approach taken in implementing the deloop section of the ray-string algorithm was a conservative one in which a brute-force and computation-intensive approach was used to explore 3D delooping. Extensive testing in 2D and 3D revealed that to eliminate loops in a fool-proof manner was not easy. In particular, deloop does not work correctly when more than a single triangle intersects another triangle, e.g. when 3 or more triangles intersect each other. Problems also arise when nodes on a triangle move in opposite directions; this is the

---

† A brute force approach requires  $0.5N^2$  intersection tests, where  $N$  is the number of triangles in the list. If  $N = 10,000$ , then 50 million pairs of triangles have to be tested for intersections! However, the use of a plane-sweep algorithm would reduce the number of intersection tests to  $N \log(N)$ . The delooping procedure could also benefit from a more intelligent approach in which only loop-prone areas are examined.



triangular analog of a pinwheeling segment. And then, in cases where the delooping does work, the integrity of the mesh is sometimes disrupted (incorrect interconnections between the triangles, segments and nodes in the mesh) and the program fails.

Given the realities of the delooping procedure, it is important to consider carefully whether or not delooping the mesh is actually necessary. The question of delooping does not come up in the string algorithm, as there is a very real need for loop removal in that algorithm. Without delooping in the string algorithm, the loops will blow up, making it impossible to proceed after some time. But in the ray-string algorithm, the loops are not dependent on the local surface of the mesh. The nodes in the mesh are moved using rays independent of each other. Therefore, the incorrect rays will hardly affect the profile, and the loops will not cause the program to self-destruct.

The loops developed during the course of the ray-string simulation do not have a serious impact on the simulation. The loops are restricted in form and shape, so they do not cross correct regions of the surface. Furthermore, the loops do not have a destructive impact on the simulation. In fact, in 3D, the simulation is more robust when the loops are not removed. The loops actually represent incorrect portions of the simulated surface, and as such, they do not truly represent the actual etched surface of the material. And since these portions of the mesh are incorrect to begin with, it is also a waste of computation time to continually trace the motion of the loops, and to store the triangles, segments and nodes that make up the loops. Unfortunately, these loops are also difficult and computationally expensive to remove. However, the loops can be easily identified visually.

So, is delooping a truly integral part of the ray-string algorithm? The loops do not affect the accuracy of the ray-string simulations, and the simulations will run without removal of the

loops. Thus the answer to the above question is a resounding "NO". But at the same time, it is desirable to remove the loops for purely aesthetic reasons. Also, certain pattern transfer processes require information of the true resist profile. In such cases, delooping is necessary.

## 8.6. 3D SIMULATION OF PHOTORESIST DEVELOPMENT

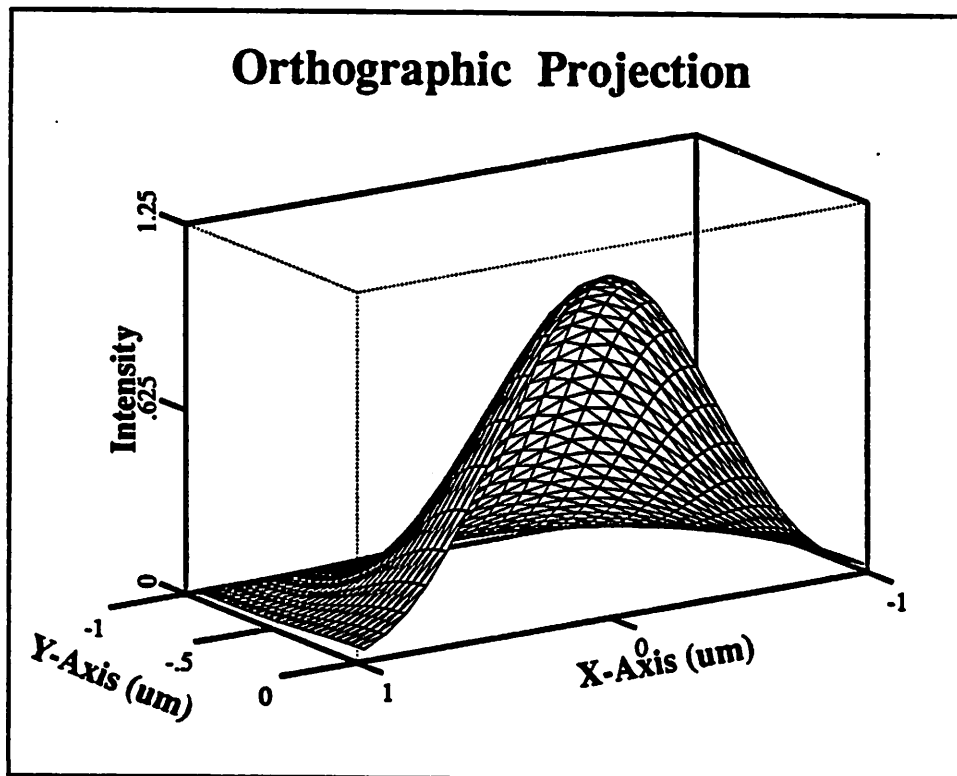
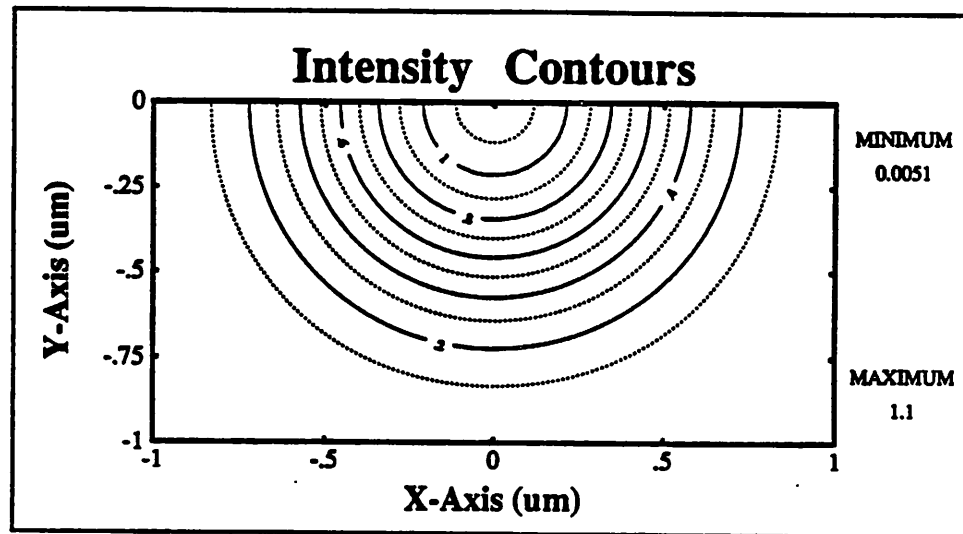
The ray-string etch simulator has been coupled to 2D imaging and exposure simulators to form the basis of a complete 3D photoresist development simulator. This simulator, named *SAMPLE-3D*, also provides display, print, and post-simulation analytical capabilities.

The flow of the 3D photolithography simulation is demonstrated in Figures 8.14 - 8.18. *SAMPLE-3D* simulates the three-dimensional (3D) profile of the developed photoresist as a function of time by first using *SPLAT* to calculate the aerial image intensity incident upon the photoresist (Figure 8.14). The exposure of the photoresist to light triggers chemical changes in the photoresist; this is modeled using *BLEACH*, a 3D exposure simulator based on Dill's<sup>1</sup> algorithm (Figure 8.15). An exposure model then generates a three-dimensional etch-rate distribution throughout the volume of the photoresist. This distribution is used in *ETCH*, a three-dimensional etch simulator, to generate a three-dimensional profile of the photoresist as a function of time (Figure 8.16).

The simulations shown in Figures 8.15 and 8.16 were run using 0.7  $\mu\text{m}$  of Olin Hunt photoresist on a bare silicon substrate. The multiple reflections in the photoresist during the exposure have resulted in a photoresist profile with standing waves. Note also that loops have formed at the fingers of the photoresist profile.

The standing waves produced by optical interference makes linewidth control difficult in

## Aerial Image Simulation



**Figure 8.14 :** Aerial image simulation of a  $1.25\ \mu\text{m}$  ( $0.8\ \lambda/\text{NA}$ ) isolated contact. The image was simulated using *SPLAT*, with  $\lambda = 0.436\ \mu\text{m}$ ,  $\text{NA} = 0.28$ , and partial coherence  $\sigma = 0.5$ .

### 3D Exposure-Bleaching Simulation

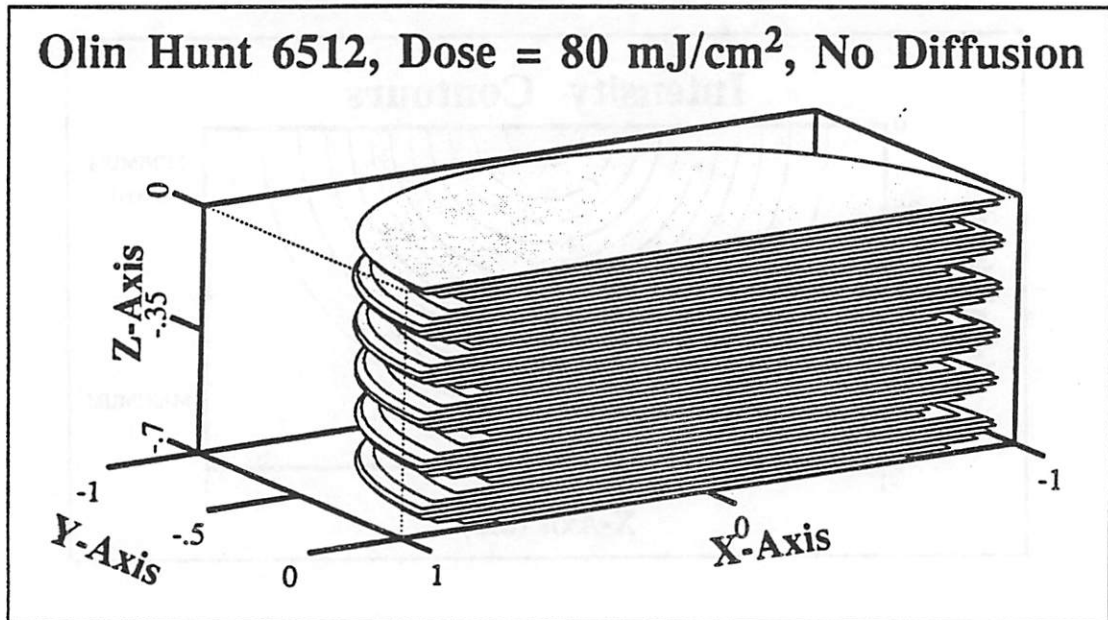


Figure 8.15: Normalized concentration of photoactive compound. The plot is a contour map of constant  $M=0.8$ .

### 3D Development-Etching Simulation

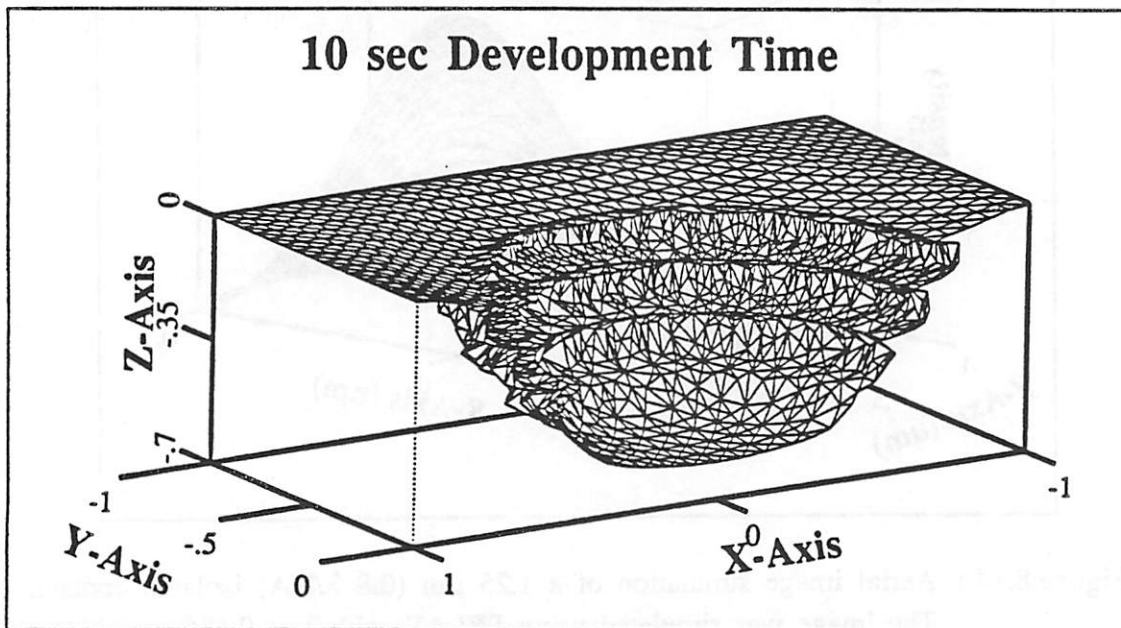


Figure 8.16: Profile of an isolated contact in  $0.7 \mu\text{m}$  Olin Hunt 6512 resist.

printing micrometer linewidths.<sup>2</sup> To reduce the extent of the standing waves, the photoresist can be baked after exposure and before development. Post-exposure bake is modeled by assuming that the chemical inhibitor diffuses from high concentration regions to low concentration regions. Figure 8.17 shows the effect of post-exposure bake on the concentration of the photoactive compound. The post-exposure bake is modeled using a one-dimensional Gaussian diffusion in the  $z$ -direction, with an average diffusion length of  $0.08\mu\text{m}$ . The standing waves in the chemical concentration have been greatly reduced. This has a significant impact on the photoresist development. As seen in Figure 8.18, the resultant photoresist profile is smooth and devoid of standing waves or loops.

In cases where loops are created, it is useful to have the capability to remove the loops. The delooping procedure described in the previous section was used to try to deloop the mesh of Figure 8.16. The result is shown in Figure 8.19. The loops have disappeared. But so has most of the mesh! This interesting result has been traced to the problem of pinwheeling triangles discussed earlier. The pinwheeling triangles have caused errors in the determination of which parts of the mesh are inside or outside the loops. As a result, most of the mesh is determined to be in error and the program goes on a mesh-deleting binge.

## 8.7. COMPUTATION ISSUES

The development-etch simulations shown in Figures 8.16 and 8.18 cost approximately 10 and 5 minutes respectively on a SUN 4/280 (0.8 MFLOPS, 10 MIPS). Approximately 2 MBytes of memory was used in the simulations; most of this amount was used to store the discrete etch-rate data. In addition, the resist-exposure simulator required approximately 8 MB of memory to calculate the photoactive compound concentration at  $50 \times 50 \times 300$  points.

### 3D Exposure-Bleaching Simulation

Olin Hunt 6512, Dose =  $80 \text{ mJ/cm}^2$ ,  $0.08 \mu\text{m}$  Diffusion

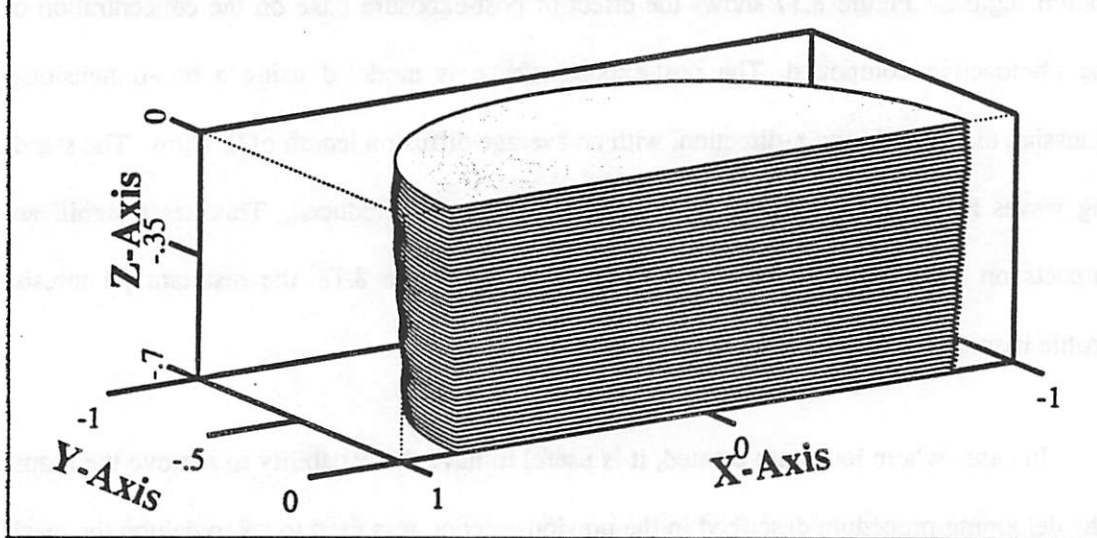


Figure 8.17: Normalized concentration of photoactive compound, after diffusion. The plot is a contour map of constant  $M=0.8$ .

### 3D Development-Etching Simulation

10 sec Development Time

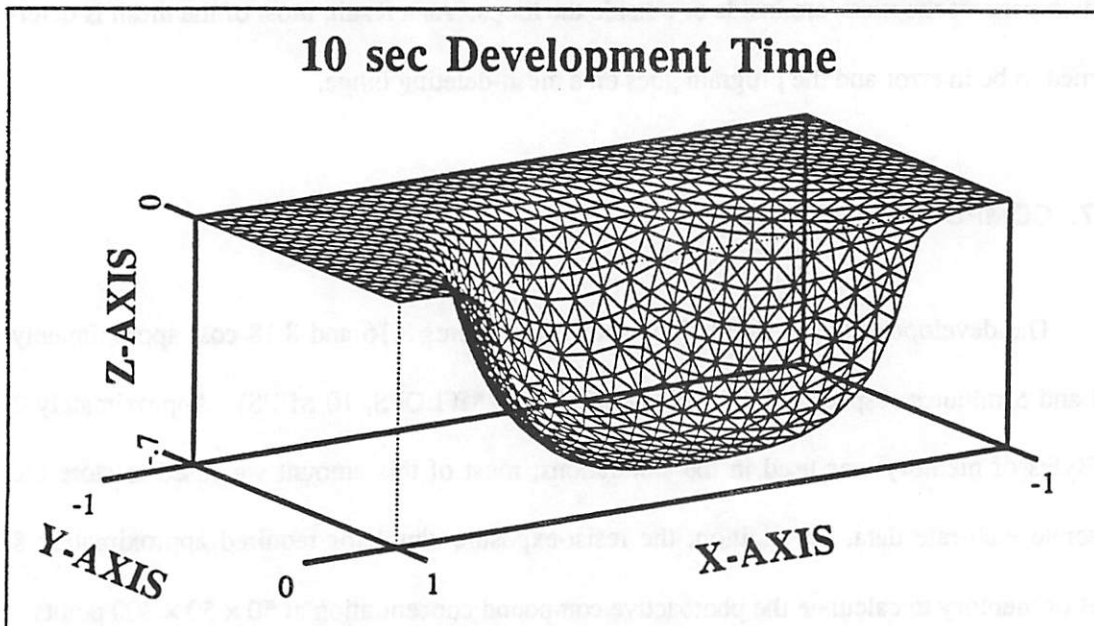
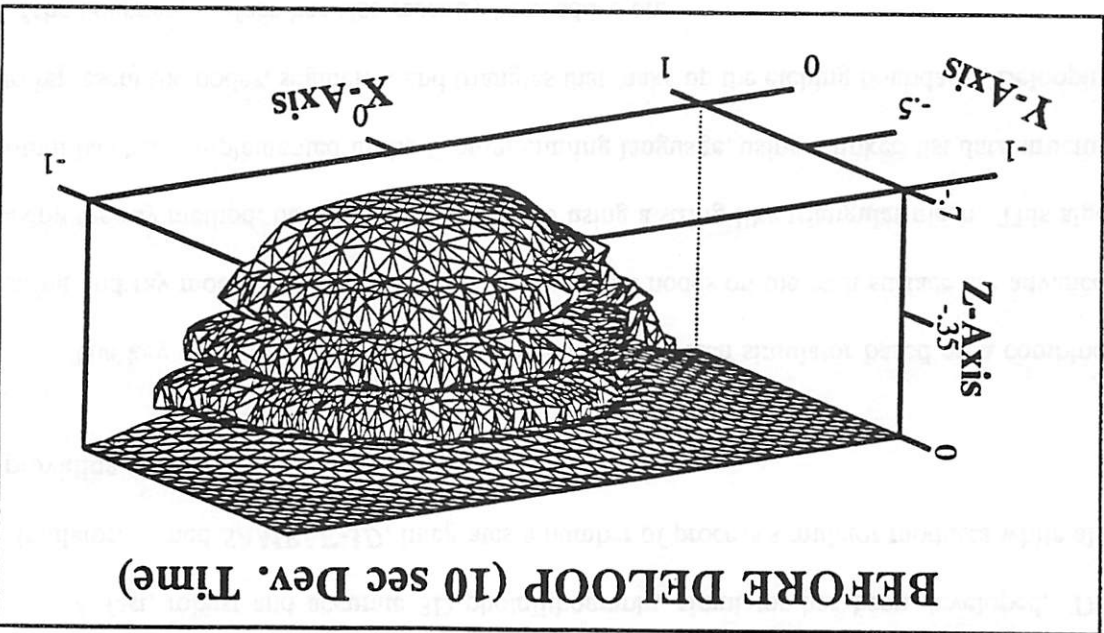
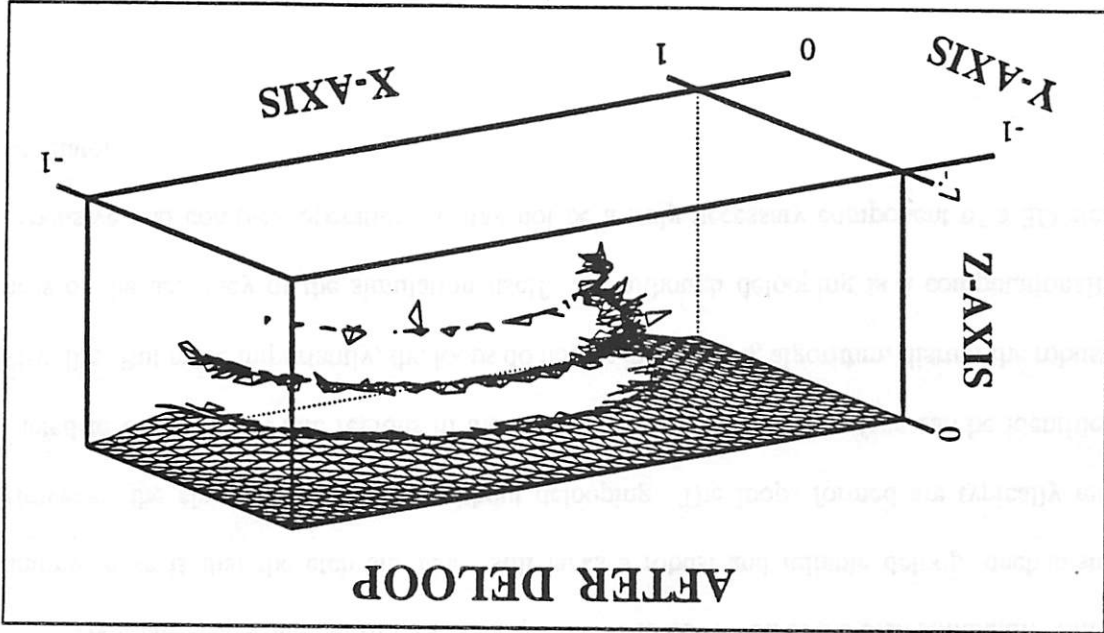


Figure 8.18: Profile of an isolated contact in  $0.7 \mu\text{m}$  Olin Hunt 6512 resist.

Figure 8.19: The effect of deloop gone awry. Most of the mesh has been deleted inadvertently due to incorrect determination of the in-loop mesh.



**3D Development-Etching Simulation**

## **8.8. SUMMARY : IMPLEMENTATION OF A 3D PHOTORESIST DEVELOPMENT SIMULATOR**

A fast, robust and accurate 3D photolithography simulator has been developed. This simulator, named *SAMPLE-3D*, integrates a number of process simulator modules while also providing display and print capabilities.

The key element of this process simulator is a 3D etch simulator based on a combined string and ray model approach. In this approach, the nodes on the etch surface are advanced using the ray method, but the nodes are joined using a string-like triangular mesh. This algorithm has been implemented in the C programming language, using a linked list data structure to represent the nodes, segments, and triangles that make up the etching boundary. Delooping of the boundary surface has also recently been added on.

There are still a number of issues and problems to be solved in the etch simulator. Chief among these is that the etch simulator still lacks a robust and reliable deloop mechanism. However, the simulation does run without delooping. The loops formed are typically restricted to the low-etch-rate regions in the material, so the true etch-surface can be identified visually. But more importantly, the loops do not, as in the string algorithm, disrupt the robustness or the accuracy of the simulation itself. So, although delooping is a computationally expensive and complex operation, it may not be a truly necessary component of a 3D etch simulator.



## REFERENCES

1. F.H. Dill, A.R. Neureuther, J.A. Tuttle, E.J. Walker, "Modeling Projection Printing of Positive Photoresists," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 456-464, July 1975.
2. E.J. Walker, "Reduction of Photoresist Standing-Wave Effects by Post-Exposure Bake," *IEEE Transactions on Electron Devices*, vol. ED-22, no. 7, pp. 464-466, July 1975.

## CHAPTER 9

### THREE-DIMENSIONAL SIMULATION OF OPTICAL LITHOGRAPHY

#### 9.1. INTRODUCTION

In submicron optical lithography, optical simulation has proved to be a useful and important technique for understanding and balancing the many complex tradeoffs between materials, exposure tools, and wafer conditions. Research to date has emphasized the application of two-dimensional aerial image simulation for investigating 2D mask-related issues in optical lithography. These issues include diverse topics such as the optical proximity effect,<sup>1</sup> defect interactions with features,<sup>2 · 3 · 4 · 5</sup> projection lens aberrations,<sup>6 · 7</sup> The 2D simulation capability has also been used to design image-monitoring test-patterns,<sup>8 · 9</sup> and to examine the resolution impact of different phase-shifting mask designs.<sup>10 · 11 · 12</sup>

Most of the aerial-image studies thus far have relied on a simple threshold interpretation of the intensity, in which a constant intensity contour is assumed to correspond to the developed photoresist profile. However, simulation and experimental studies<sup>4 · 8 · 13</sup> have shown that in certain mask configurations, nonvertical resist dissolution effects could cause discrepancies between the simulated intensity contours and the experimentally printed features. In such situations, it is necessary to resort to a full rigorous 3D simulation to determine the resist profiles and thus understand the resist development process. The discrepancies caused by nonlinear and nonvertical resist effects has been the major driving force behind the development of a complete 3D photolithography simulator. Now that a fast and complete 3D resist development simulator has been developed, it is possible to study and model the effect of nonlinear and nonvertical resist dissolution on the projection-printed patterns.

## 9.2. NON-VERTICAL RESIST DISSOLUTION EFFECTS

### 9.2.1. The Intensity Threshold Model

The intensity threshold model has proved to be a useful tool for studying the issues associated with printing 2D mask patterns. This approach avoids a rigorous and time-consuming 3D resist development simulation in favor of a simpler threshold interpretation of the intensity contours to determine the shape of the final developed profile.

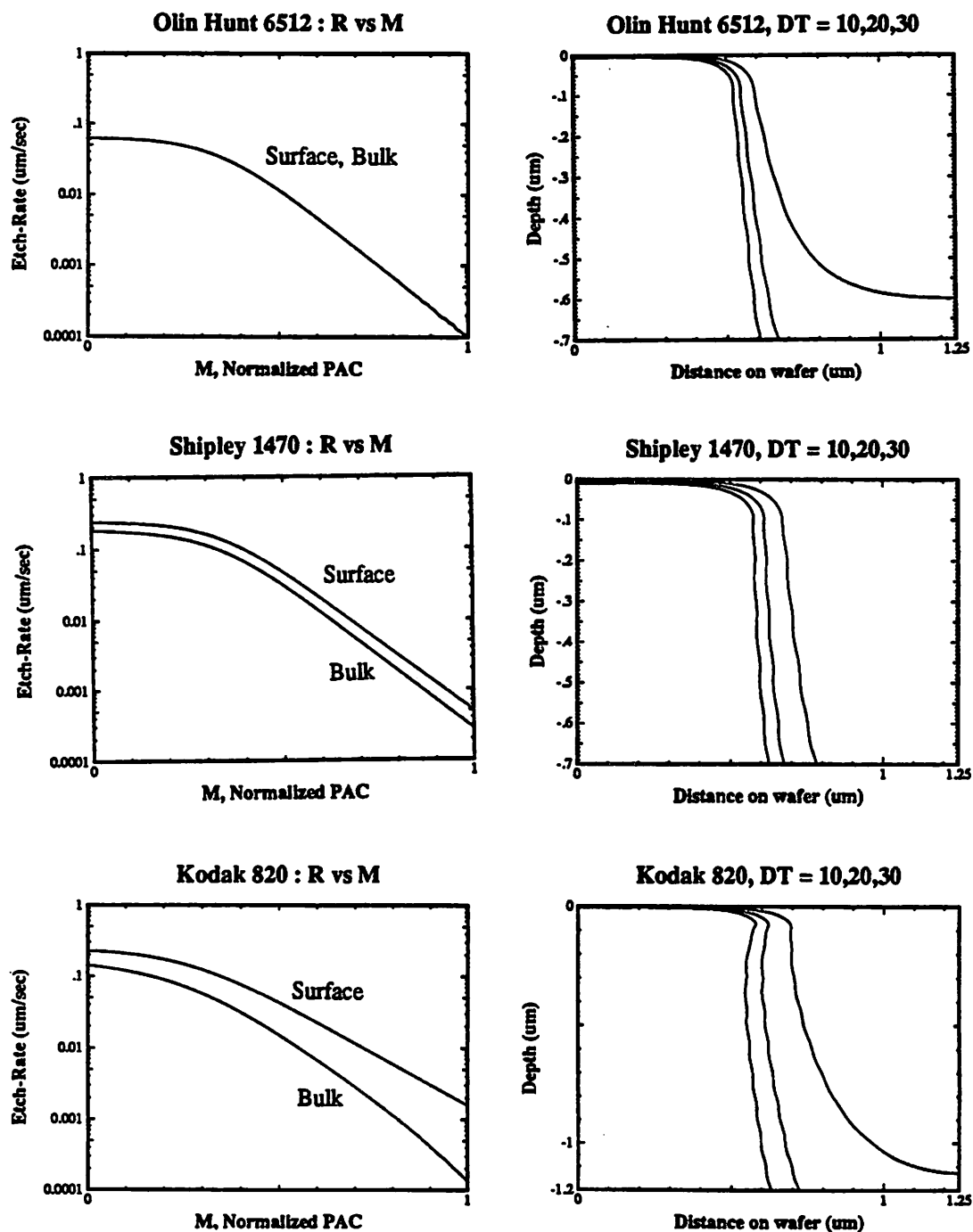
*SAMPLE-3D* has been used to investigate the correlation between the 2D aerial image and the 3D developed resist profile. Simulations have been carried out on three different positive photoresists, Shipley 1470, Olin Hunt 6512, and Kodak 820. Shipley 1470 represents a prototypical medium-contrast photoresist which was used heavily in the 1970's. Lately, a number of newer improved photoresists with higher contrast have been introduced. For the purposes of simulation, Olin Hunt 6512 was selected as an example of a high-contrast positive photoresist. Resists with surface-rate retardation, such as Kodak 820, are also frequently used to print patterns with square profiles; surface-rate retardation decreases the top-loss in the unexposed regions and results in better wall-angles. The simulation parameters for each of these photoresists is provided in Table 9.1. In addition, the *SAMPLE*-simulated resist profiles of  $0.8 \lambda/NA$  lines and spaces printed on the three different photoresists are plotted in Figure 9.1. Also plotted in Figure 9.1 are the etch-rate (R) vs photoactive compound concentration (M) for the three different photoresists.

Figure 9.2 shows the aerial image and 3D resist profile simulations of a  $1.25 \mu\text{m}$  ( $0.8 \lambda/NA$ ) isolated transparent elbow in a dark field mask. The aerial image simulation in this case used an exposure wavelength  $\lambda$  of  $0.436 \mu\text{m}$ , numerical aperture NA of 0.28, and partial

Olin Hunt 6512 resist system
Projection System : $\lambda = 0.436 \mu\text{m}$ , $\text{NA} = 0.28$ , $\sigma = 0.5$
Exposure : $A = 0.640 \mu\text{m}^{-1}$ , $B = 0.040 \mu\text{m}^{-1}$ , $C = 0.010 \text{cm}^2/\text{mJ}$ Post-Exposure Bake Diffusion = $0.08 \mu\text{m}$ Best Exposure Dose = $240 \text{mJ}/\text{cm}^2$ Refractive Index = 1.68 Resist Thickness = $0.70 \mu\text{m}$ Substrate : Si ( $n = 4.73-j0.14$ )
Development (Kim model) $R1 = 0.062 \mu\text{m}/\text{s}$ , $R2 = 0.0001 \mu\text{m}/\text{s}$ , $R3 = 8.5$
Shipley Microposit 1470 resist system
Projection System : $\lambda = 0.436 \mu\text{m}$ , $\text{NA} = 0.28$ , $\sigma = 0.5$
Exposure : $A = 0.580 \mu\text{m}^{-1}$ , $B = 0.030 \mu\text{m}^{-1}$ , $C = 0.014 \text{cm}^2/\text{mJ}$ Post-Exposure Bake Diffusion = $0.08 \mu\text{m}$ Best Exposure Dose = $80 \text{mJ}/\text{cm}^2$ Refractive Index = 1.68 Resist Thickness = $0.70 \mu\text{m}$ Substrate : Si ( $n = 4.73-j0.14$ )
Development (Kim model) $R1 = 0.24 \mu\text{m}/\text{s}$ , $R2 = 0.0005 \mu\text{m}/\text{s}$ , $R3 = 8.1$ $R4 = 0.24 \mu\text{m}$ , $R5 = 0.76$ , $R6 = 0.55$
KODAK 820 resist system
Projection System : $\lambda = 0.436 \mu\text{m}$ , $\text{NA} = 0.28$ , $\sigma = 0.5$
Exposure : $A = 0.510 \mu\text{m}^{-1}$ , $B = 0.031 \mu\text{m}^{-1}$ , $C = 0.013 \text{cm}^2/\text{mJ}$ Post-Exposure Bake Diffusion = $0.08 \mu\text{m}$ Best Exposure Dose = $110 \text{mJ}/\text{cm}^2$ Refractive Index = 1.68 Resist Thickness = $1.20 \mu\text{m}$ Substrate : Si ( $n = 4.73-j0.14$ )
Development (Kim model) $R1 = 0.23 \mu\text{m}/\text{s}$ , $R2 = 0.0016 \mu\text{m}/\text{s}$ , $R3 = 5.6$ $R4 = 0.25 \mu\text{m}$ , $R5 = 0.62$ , $R6 = 0.08$

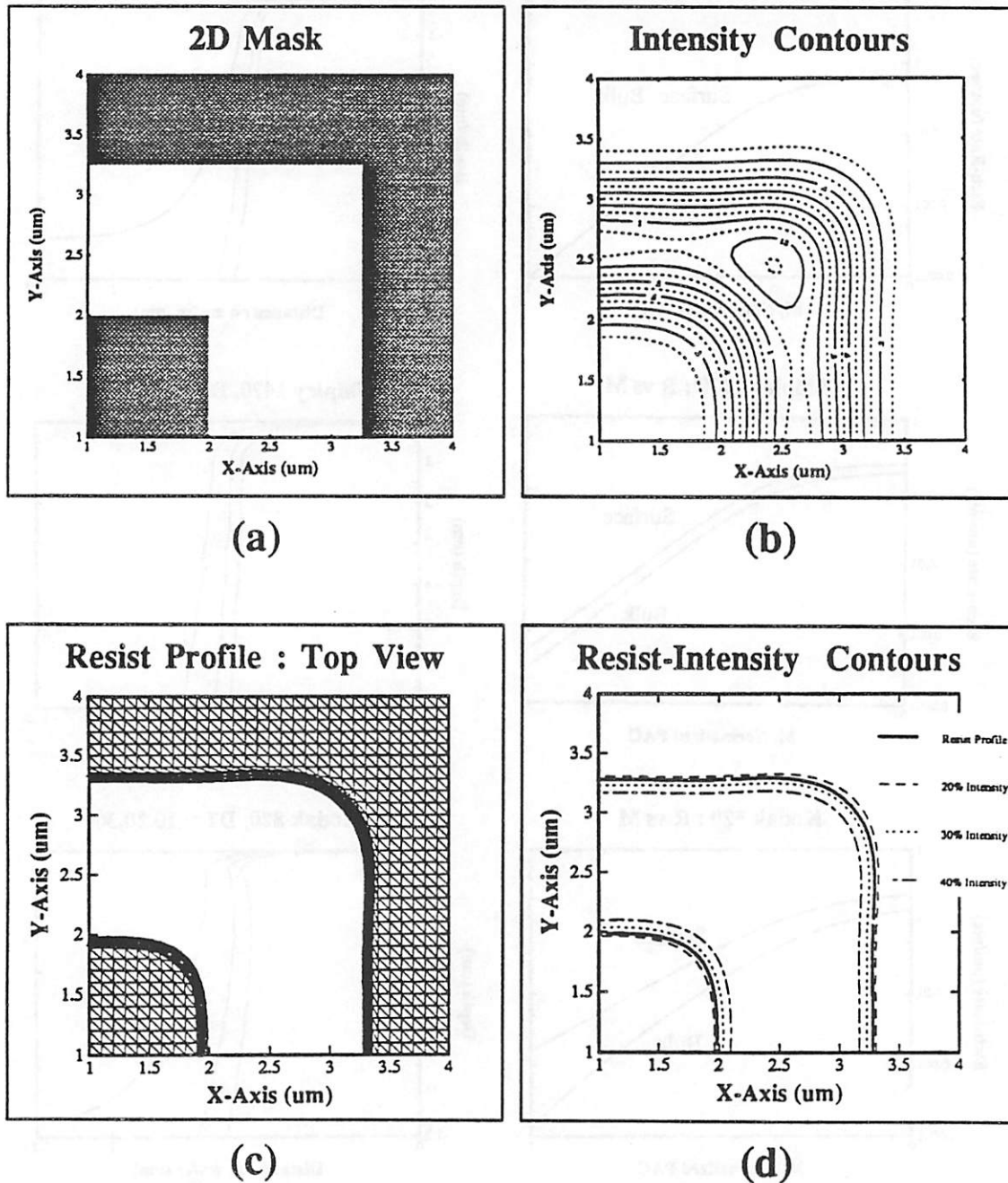
Table 9.1 : Photoresist/Substrate Parameters

# PHOTORESIST SYSTEMS



**Figure 9.1 :** Photoresist systems : Etch-rate vs photoactive compound concentration and *SAMPLE* resist profiles for 3 different photoresists. The dose was adjusted so that 1.25  $\mu\text{m}$  ( $0.8 \lambda/\text{NA}$ ) lines and spaces printed 1:1 with 30 seconds of development time.

**1.25  $\mu\text{m}$  (0.8  $\lambda/\text{NA}$ ) Isolated Transparent Elbow**  
**Olin Hunt 6512 : Dev. Time =30 sec, Dose =240  $\text{mJ}/\text{cm}^2$**



**Figure 9.2 :** Demonstration of the intensity threshold model. The resist profile, simulated on 0.7  $\mu\text{m}$  of Olin Hunt 6512 resist, agrees with the 30% intensity contour. Thus the intensity contour is sufficient for predicting the linewidth of the developed profile.

coherence  $\sigma = 0.5$ . The image was then passed on to the 3D resist-exposure simulation for exposure (with  $0.08 \mu\text{m}$  post-exposure-bake diffusion) on  $0.7 \mu\text{m}$  of Olin Hunt 6512 resist. The resultant etch-rate distribution was then used to calculate the profile after 30 seconds of development time.

The aerial image contours and the developed resist profile are plotted on top of each other in Figure 9.2d. The dark solid line, which represents the resist profile, is very similar to the 20–40% intensity contour plots. This means that in this particular simulation, it is sufficient to use a constant intensity contour to model the opening in the photoresist. Similar simulations on Shipley 1470 and Kodak 820 also bear out this result.

In the elbow configuration of Figure 9.2, the dissolution or etch-front starts out from the center of the elbow and sweeps towards the outside edges of the elbow. This is essentially a diverging etch in which the development of the resist is dominated primarily by the dissolution action from the center of the elbow. Furthermore, the intensity throughout the elbow remains high along the elbow, so the development speed is relatively constant along the elbow. As a result, the resist development of the elbow is essentially a linear process and can be described using the intensity threshold model. There has been, however, some experimental evidence that the intensity threshold model does not hold true in all cases.<sup>4, 8</sup> The deviations from the intensity threshold model and their causes will be discussed in the following sections.

### 9.2.2. A Centered Opaque Defect In a Line-Space Array

Figure 9.3 shows SEM photographs of a  $0.5 \mu\text{m}$  opaque defect placed at the center of a transparent space in a  $1.3 \mu\text{m}$  line-space array. The patterns were printed on  $1.2 \mu\text{m}$  of Kodak

**Square Opaque Defect between Opaque Lines**

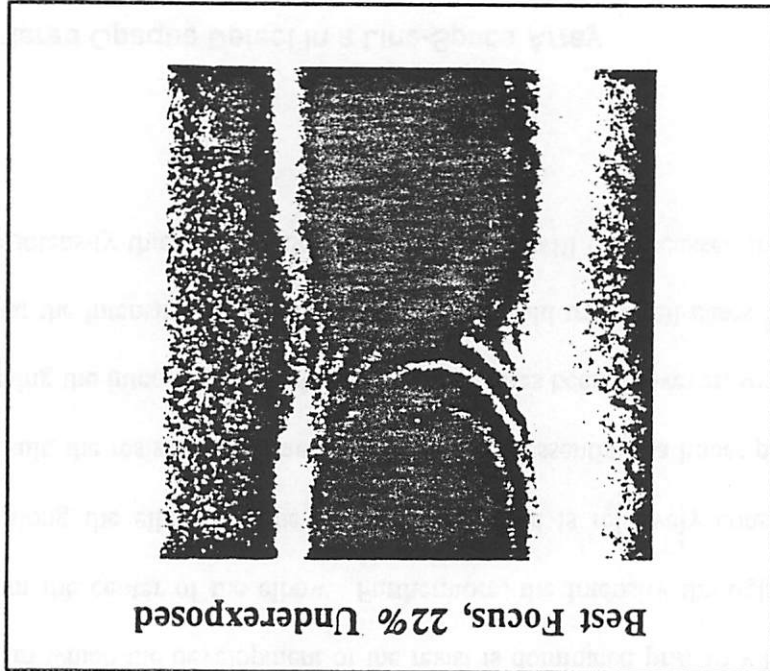
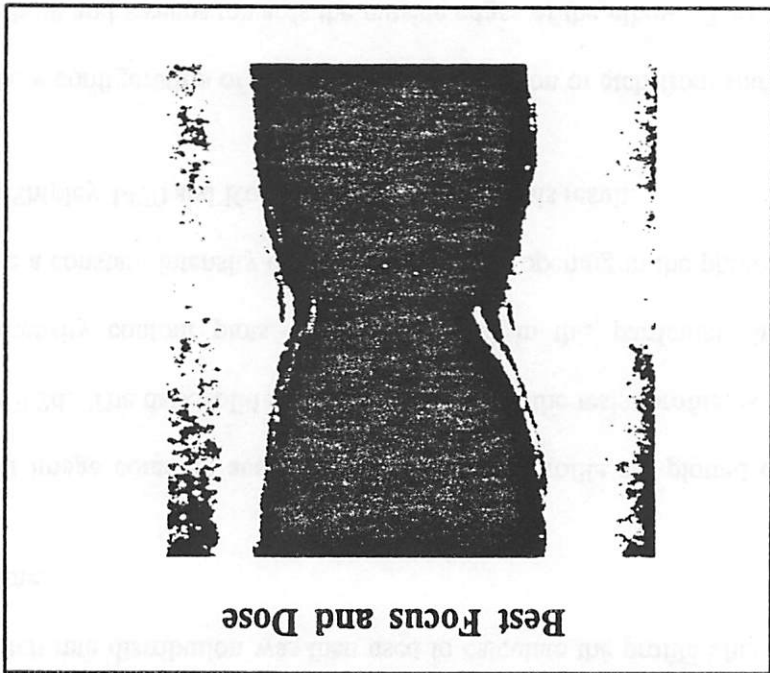


Figure 9.3 : SEM photographs of a 0.5  $\mu\text{m}$  opaque centered defect, printed on 1.2  $\mu\text{m}$  of Kodak 820 positive photoresist on a Si substrate.

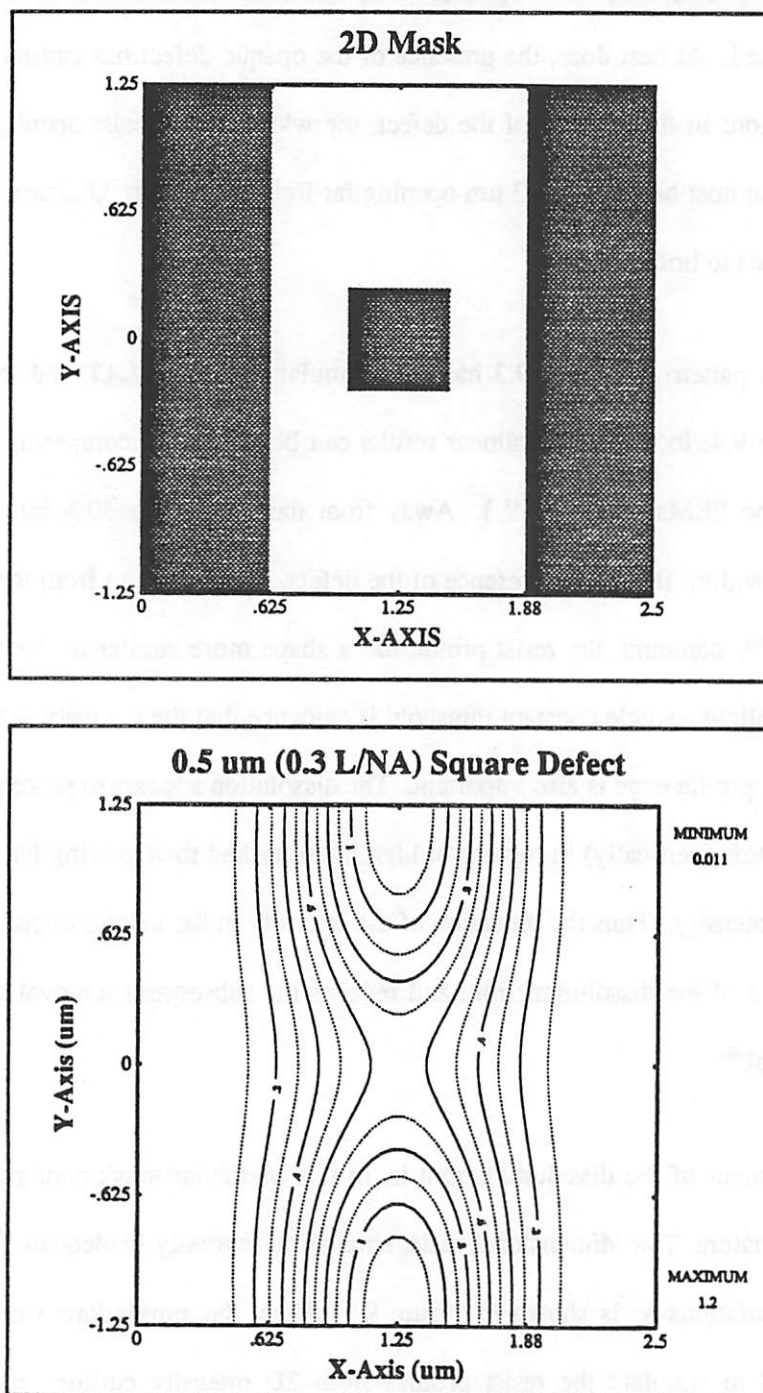


820 positive resist on a Si substrate, using a GCA g-line ( $\lambda = 0.436 \mu\text{m}$ ) stepper with NA of 0.28. The two photographs show the resist profile at best focus and dose, and at best focus and 22% underexposed. At best dose, the presence of the opaque defect has caused a significant linewidth variation; in the vicinity of the defect, the width of the resist opening is approximately  $0.7 \mu\text{m}$ , almost half of the  $1.3 \mu\text{m}$  opening far from the defect. Underexposure causes the two resist lines to bridge.

The defect pattern in Figure 9.3 has been simulated with *SPLAT* and the results are shown in Figure 9.4. Interesting nonlinear results can be seen from comparing the intensity contours with the SEMs of Figure 9.3. Away from the defect, the 30% intensity contour predicts the linewidth. But in the presence of the defect, the protrusion from the line reaches out to the 40-60% contours; the resist profile has a shape more similar to the 50% contour. This failure to follow a single constant threshold is evidence that the intensity in the neighborhood of the final profile edge is also important. The dissolution appears to proceed initially by moving downwards (vertically) in regions of high intensity and then moving laterally into the regions of low intensity. Thus the reduction of the intensity in the neighborhood of the defect delays the arrival of the dissolution front and reduces the subsequent removal of the foot of the resist material.<sup>4</sup>

The movement of the dissolution front is, in this particular mask configuration, three-dimensional in nature. Two-dimensional resist simulation is totally inadequate for predicting the linewidth variations as is shown in Figure 9.5. Here, the simulations were carried out using *SAMPLE* to simulate the resist profiles from 2D intensity cutlines calculated with *SPLAT*. The *SAMPLE* simulations were run on  $0.7 \mu\text{m}$  of Shipley 1470 positive photoresist, and the profiles are shown at 5, 10, 15 and 20 seconds of development time. The 15 second developed profile is very interesting. Using a north-south intensity cut, the simulation predicts

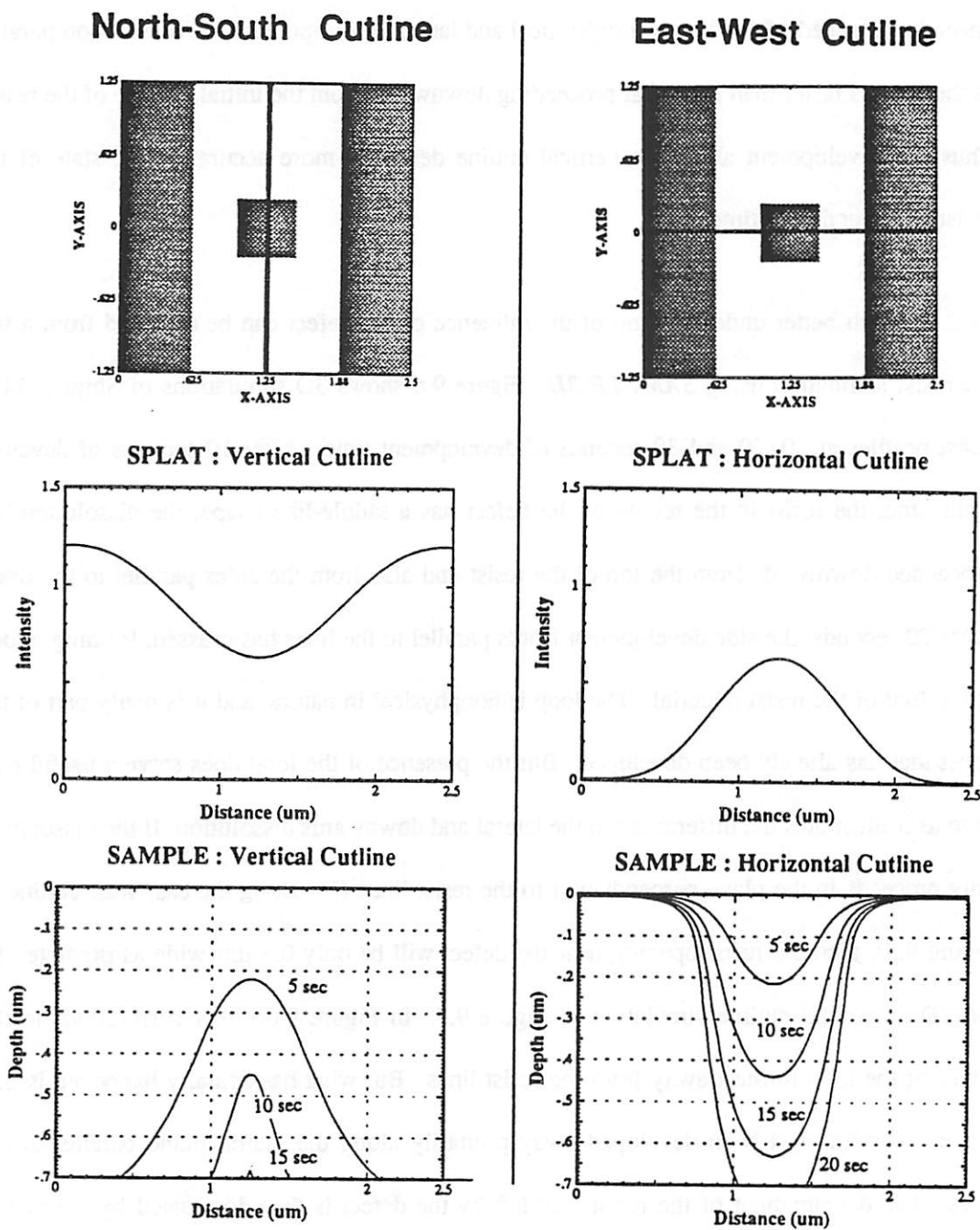
## Square Opaque Defect between Opaque Lines



**Figure 9.4:** Image intensity contour plot of a  $0.5 \mu\text{m}$  opaque square defect in a  $1.3 \mu\text{m}$  equal line-space array. The simulation was run using  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA}=0.28$  and partial coherence  $\sigma = 0.5$ .

## Resist Simulation with Cutlines

Shipley 1470 : Dev. Time = 5-20 sec, Dose = 80 mJ/cm<sup>2</sup>



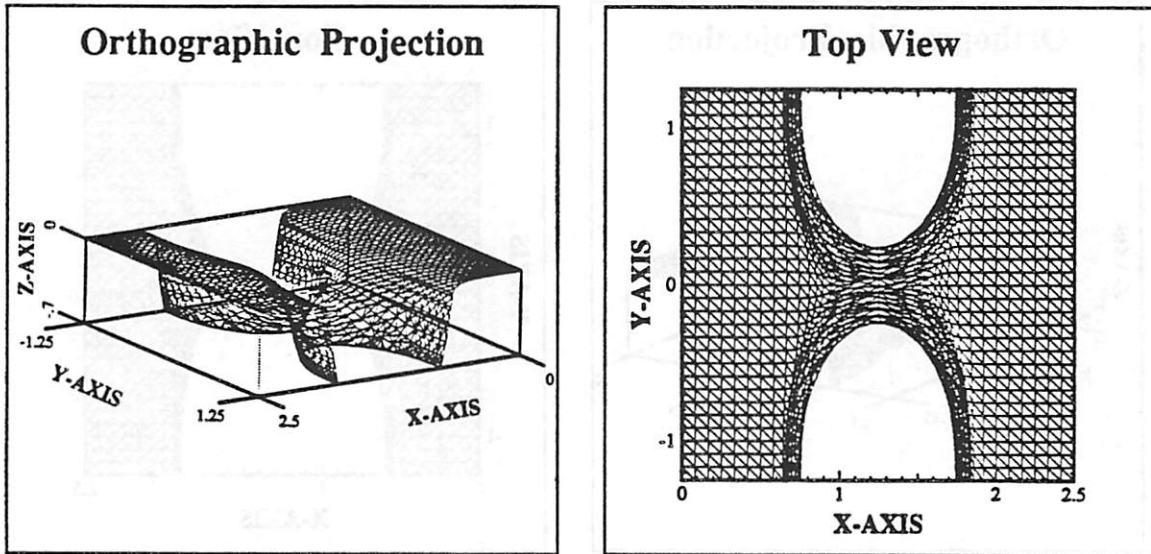
**Figure 9.5:** Resist Simulation with Cutlines. The north-south and east-west intensity cutlines result in different remaining resist heights at the center of the defect after 15 sec of development.

that the resist will be almost completely developed by 15 seconds. But at the very same location at the center of the defect, the east-west cutline simulation predicts a remaining resist height of almost  $0.1\ \mu\text{m}$ . The difference between the two sets of simulations is due to differences in the speed of the downwards/vertical and lateral development. The dissolution parallel to the lines is faster than from that proceeding downwards from the initial surface of the resist. Thus the development along the vertical cutline describes more accurately the state of the resist as a function of time.

A much better understanding of the influence of the defect can be obtained from a full 3D resist simulation using *SAMPLE-3D*. Figure 9.6 shows 3D simulations of Shipley 1470 resist profiles at 10, 20 and 30 seconds of development time. After 10 seconds of development time, the resist in the region of the defect has a saddle-like shape; the dissolution has proceeded downwards from the top of the resist and also from the sides parallel to the lines. After 20 seconds, the side development fronts parallel to the lines has crossed, forming a loop at the foot of the resist material. The loop is nonphysical in nature, and it is really part of the resist that has already been developed. But the presence of the loop does serve a useful purpose as it illustrates the differences in the lateral and downwards dissolution. If the dissolution only proceeds in the plane perpendicular to the resist lines (i.e. along the east-west cutline in Figure 9.5), then the resist opening near the defect will be only  $0.4\ \mu\text{m}$  wide as predicted by the 2D horizontal cutline simulation in Figure 9.5. In Figure 9.6b, this corresponds to the edges of the loop furthest away from the resist lines. But what has actually happened is that the resist saddle has been developed away primarily along the lateral plane parallel to the lines. The development of the resist foot left by the defect is thus dominated by the lateral north-south dissolution, and the foot develops faster than predicted in the east-west cutline simulation.

## Shipley 1470 : Best Exposure (Dose=80mJ/cm<sup>2</sup>)

(a) Development Time = 10 seconds



(b) Development Time = 20 seconds

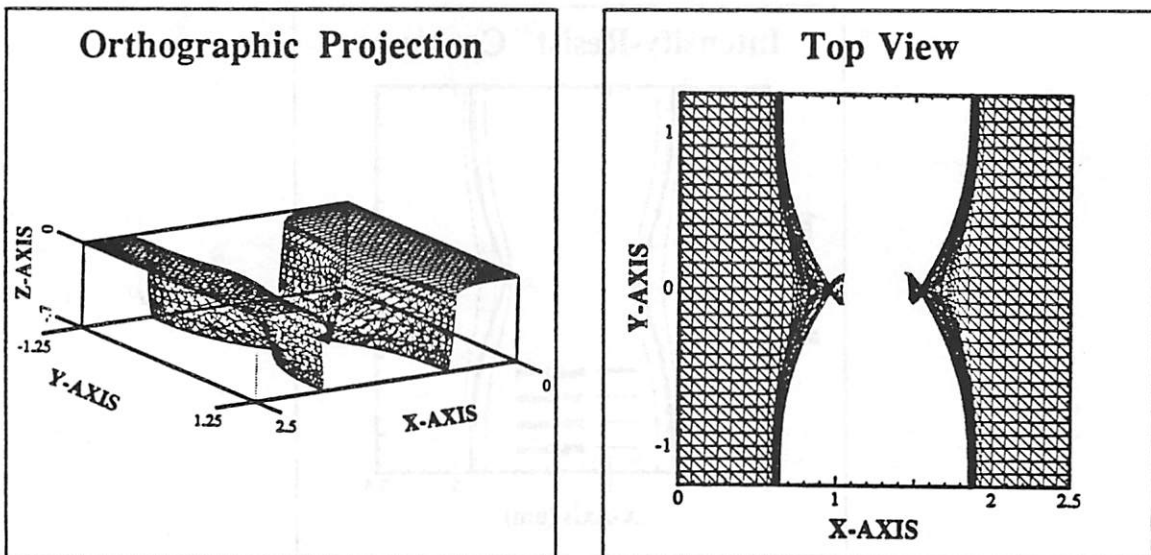
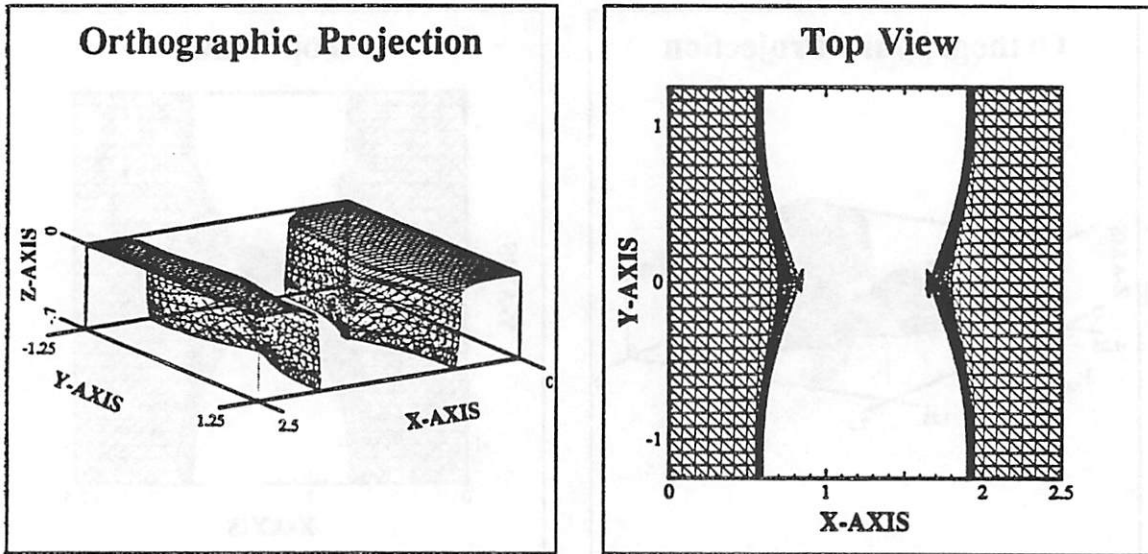


Figure 9.6 : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Shipley 1470 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

## Shibley 1470 : Best Exposure (Dose=80 mJ/cm<sup>2</sup>)

(c) Development Time = 30 seconds



(d) Resist Profile vs Intensity Contours

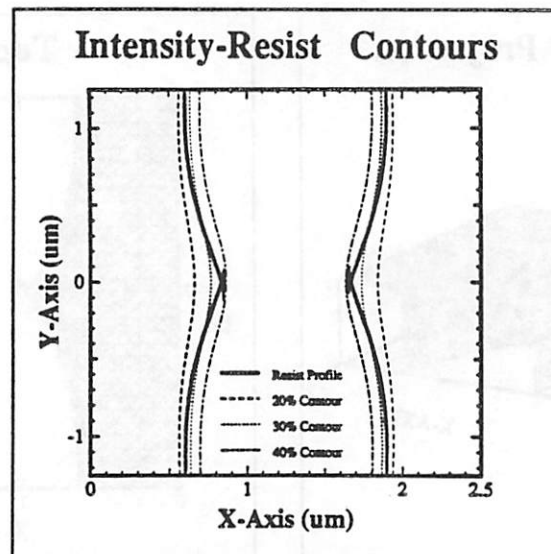


Figure 9.6 (cont) : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Shibley 1470 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

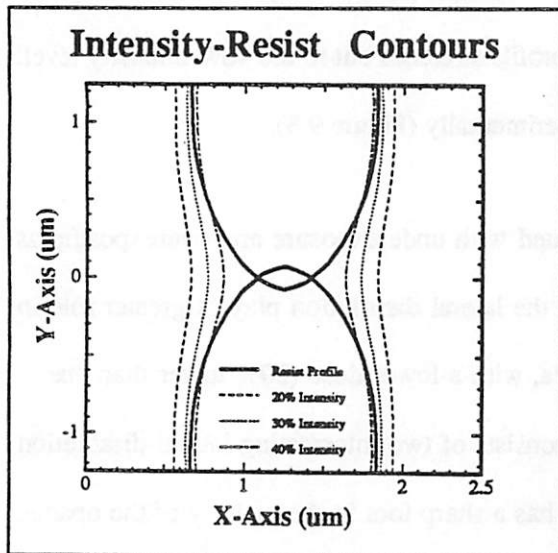
After 30 seconds of development, the resist at the center of the defect has been completely developed away, leaving a protruding foot at the sides of the lines. In Figure 9.6d, the 30-second developed profile at the Si substrate is plotted against the 20-40% intensity contours from Figure 9.4. The resist profile corresponds to the 30% intensity contour far from the defect, but in the region of the defect, the resist profile stretches out to the 40% intensity level. This is very similar to the situation observed experimentally (Figure 9.3).

The 3D resist profile has also been simulated with underexposure and overexposure as shown in Figure 9.7. At a lower exposure dose, the lateral dissolution plays a greater role in the resist development. As shown in Figure 9.7a, with a lower dose (20% lower than "best" exposure), the resist profile at the Si substrate consists of two intersecting lateral dissolution fronts. The true resist profile (outside the loop) has a sharp foot in the vicinity of the opaque defect. The protrusion due to the defect is, however, reduced at higher exposure doses. This effect is due to the dependence of the etch-rates on the exposure dose. At a higher exposure dose, the dissolution speed in the downwards direction is increased, and thus the lateral development parallel to the lines is almost canceled out.

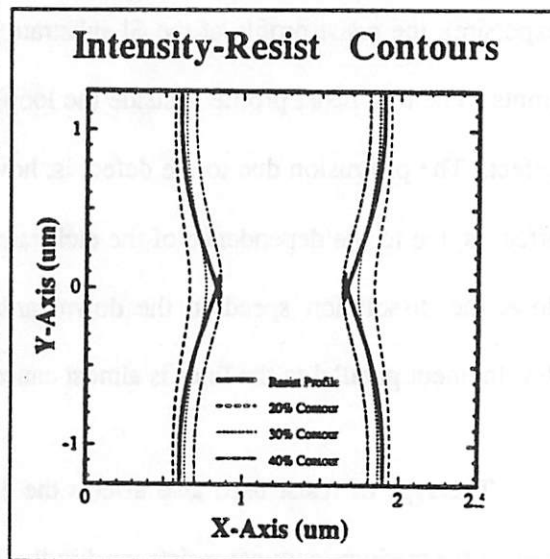
The type of resist used also affects the development process. The previous results are typical for medium-contrast resists used in the late 70's and early 80's. Improved resists with higher contrasts such as Olin Hunt 6512 will reduce the printability of defects. Figure 9.8 shows the 3D simulated patterns printed on 0.7  $\mu\text{m}$  of Olin Hunt resist, at best exposure, 20% underexposed, and 20% overexposed. Olin Hunt 6512 is a high-contrast resist, so the development etch-rate is relatively constant as long as the intensity remains above a certain intensity level. In the opaque defect mask configuration, the intensity at the center of the defect is still high enough so that the downwards etch speed in the vicinity of the defect is approximately the same as that far from the defect. As a result, there is little or no lateral

## Shipley 1470 : 30 sec Development Time

- (a) Underexposed  
(Dose = 64 mJ/cm<sup>2</sup>)



- (b) Best Exposure  
(Dose = 80 mJ/cm<sup>2</sup>)



- (c) Overexposed  
(Dose = 96 mJ/cm<sup>2</sup>)

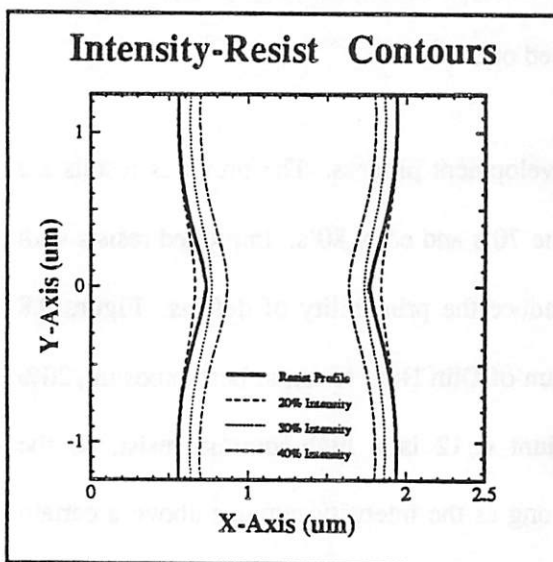
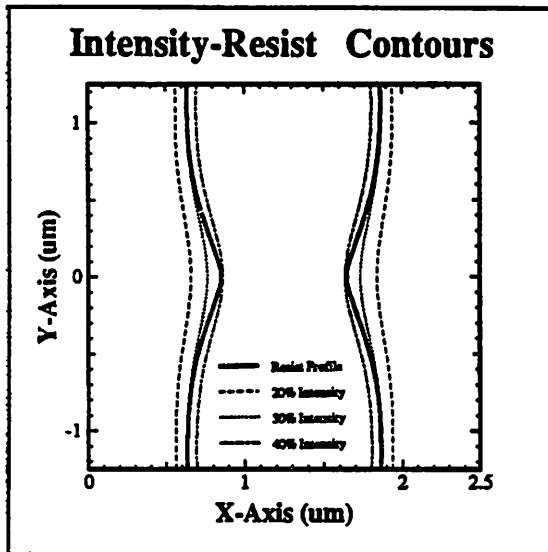


Figure 9.7: 3D Resist Development simulated on Shipley 1470 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

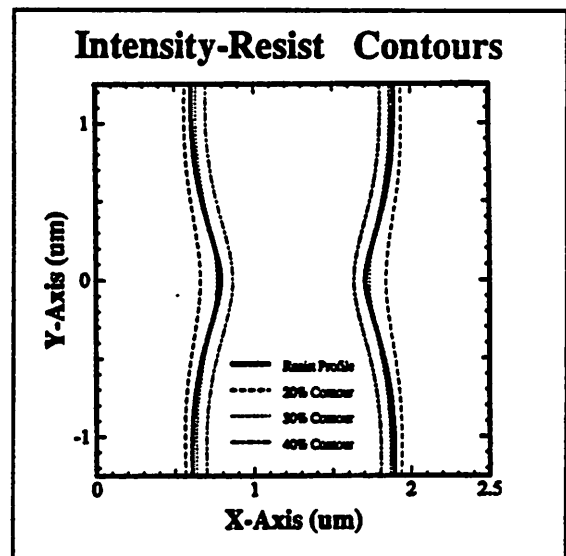


## Olin Hunt 6512 : 30 sec Development Time

(a) Underexposed  
(Dose = 192 mJ/cm<sup>2</sup>)



(b) Best Exposure  
(Dose = 240 mJ/cm<sup>2</sup>)



(c) Overexposed  
(Dose = 288 mJ/cm<sup>2</sup>)

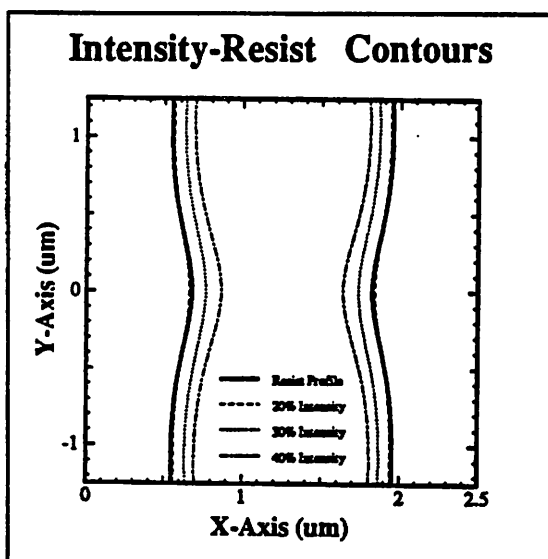


Figure 9.8: 3D Resist Development simulated on Olin Hunt 6512 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

development parallel to the resist lines. However, the lowering of the intensity at the defect center still slows down the development fronts and reduces the subsequent removal of the foot of the resist material.

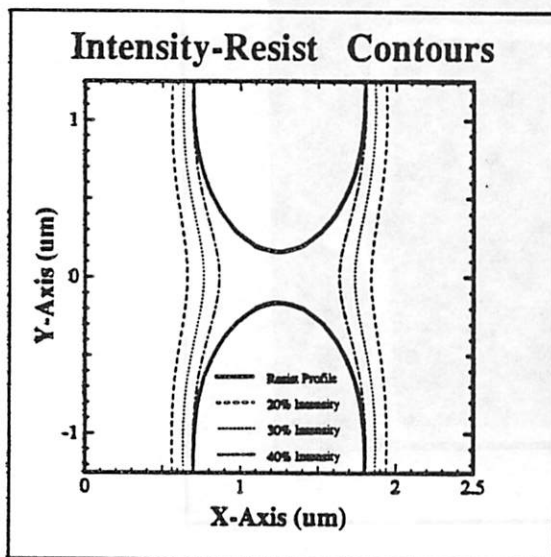
The simulations also indicate that the use of surface-rate retardation increases the sensitivity of the printed pattern to defects. Figure 9.9 shows the 3D simulated profiles printed on Kodak 820 photoresist. When the resist is underexposed by 20%, the resist patterns bridge. At "best" exposure of  $110 \text{ mJ/cm}^2$ , the lateral etch-fronts have crossed, so there is a sharp foot at the edges of the resist near the defect. This simulated result, as well as that of the underexposed simulation, is in excellent agreement with the experimental data shown in the SEMs of Figure 9.3. Overexposure by 20% decreases the size of the foot, but the protrusion caused by the defect is still quite significant. With surface-rate retardation, the role of lateral dissolution in the development of the Kodak 820 resist has increased, thus increasing the sensitivity of this resist to small opaque defects.

### 9.2.3. An Opaque Defect In a Corner

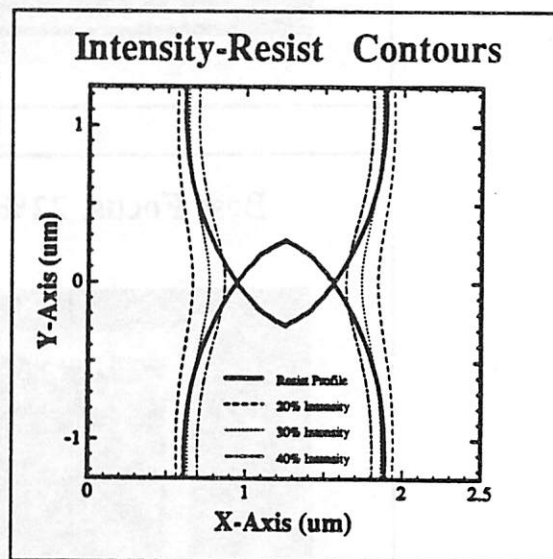
Nonvertical resist development effects have also been observed in situations where an opaque defect is placed at a corner of a large opaque square.<sup>4</sup> Figure 9.10 shows an opaque square area with  $0.5 \text{ }\mu\text{m}$  opaque defects located diagonally from the corners. These corner defects do not print except when underexposed. Figure 9.11 shows the aerial image simulations corresponding to the upper left corner of this configuration. The opaque defect is placed  $0.2 \text{ }\mu\text{m}$  from the corner of the large opaque square. At best dose, the resist profile follows approximately the 30% intensity contour. But underexposure by 22% as shown in Figure 9.11b does not follow the 38% intensity contour as would be expected from the constant intensity threshold model. Instead, the large area of 60% intensity does not clear completely

## Kodak 820 : 30 sec Development Time

(a) Underexposed  
(Dose = 88 mJ/cm<sup>2</sup>)



(b) Best Exposure  
(Dose = 110 mJ/cm<sup>2</sup>)



(c) Overexposed  
(Dose = 132 mJ/cm<sup>2</sup>)

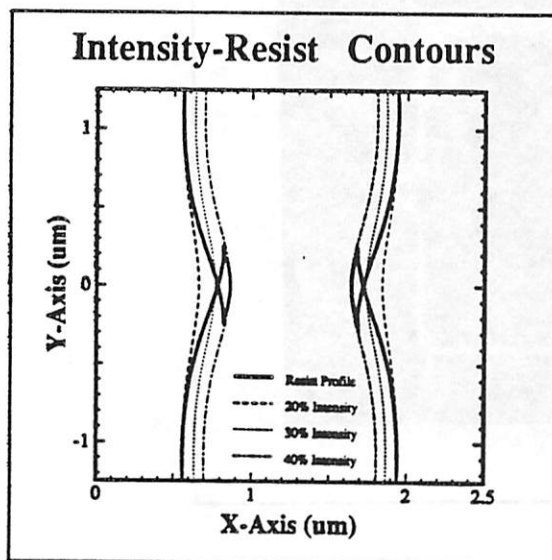


Figure 9.9: 3D Resist Development simulated on Kodak 820 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

## Square Opaque Defect in a Corner

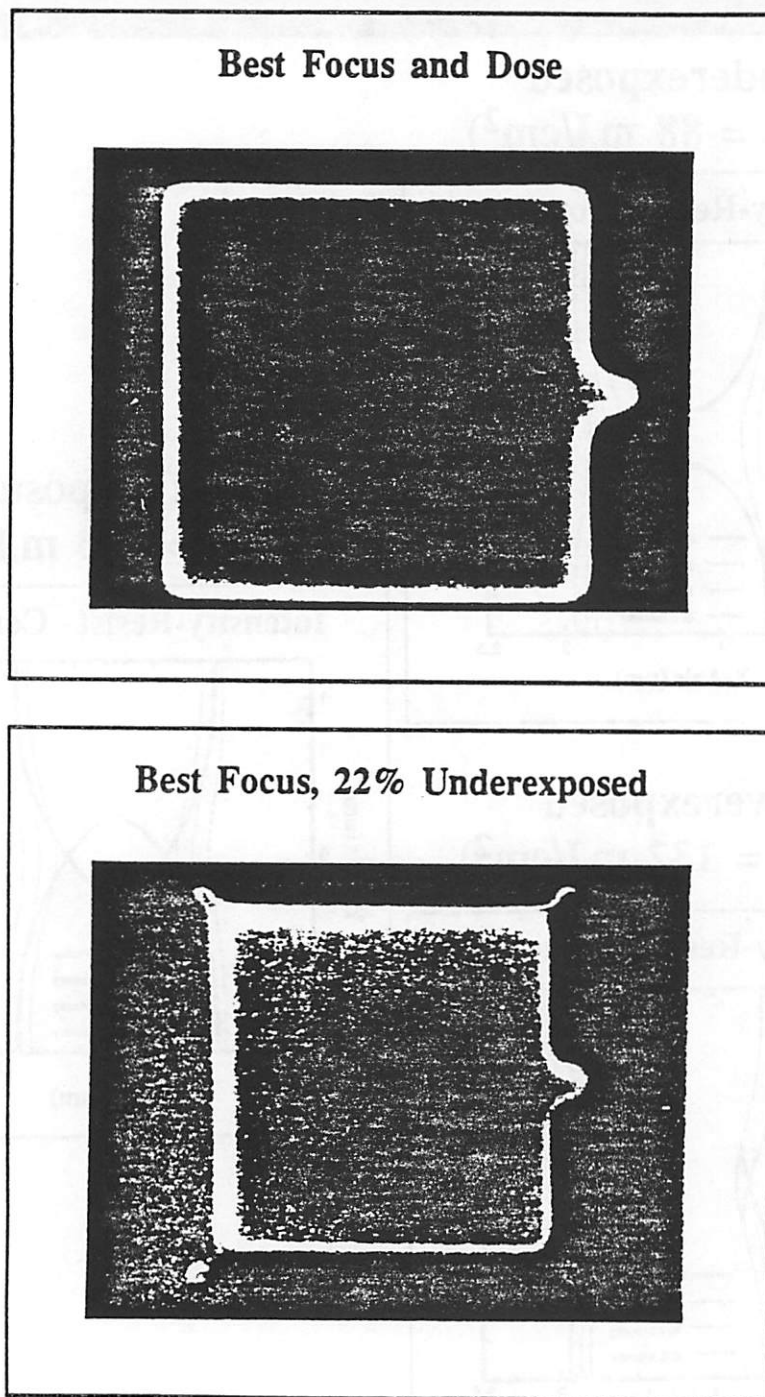
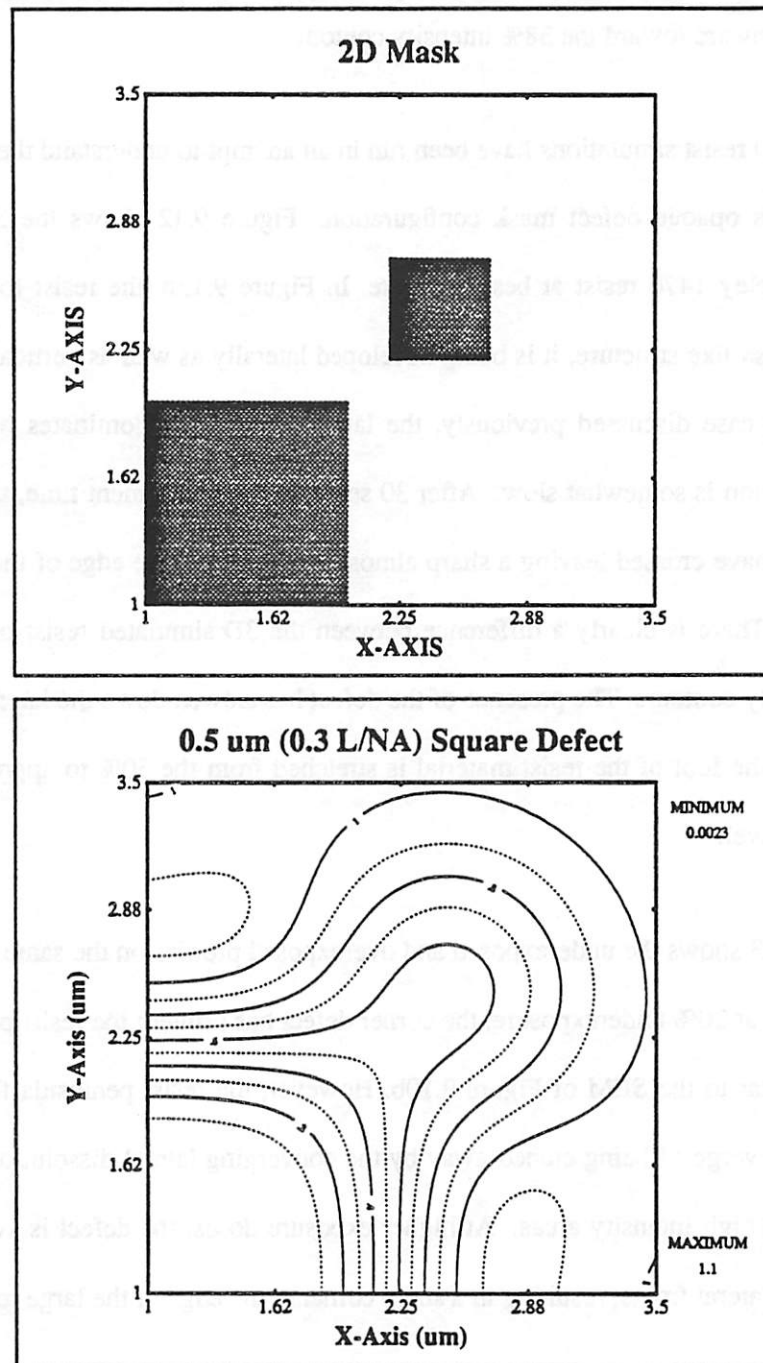


Figure 9.10 : SEM photographs of a  $0.5\ \mu\text{m}$  opaque defect at the diagonal corners of a large opaque square. The patterns were printed on  $1.2\ \mu\text{m}$  of Kodak 820 positive photoresist on a Si substrate.

## Square Opaque Defect in a Corner



**Figure 9.11:** Image intensity contour plot of a  $0.5\ \mu\text{m}$  opaque square defect placed  $0.2\ \mu\text{m}$  from the corner of a large square. The simulation, which covers the left corner of the large square in Fig 9.10, was run using  $\lambda = 0.436\ \mu\text{m}$ ,  $\text{NA} = 0.28$  and partial coherence  $\sigma = 0.5$ .

giving rise to the printed defect. Here it appears that with underexposure the 60% exposure region no longer clears through vertical dissolution. Thus there is no chance for lateral dissolution to work inward toward the 38% intensity contour.

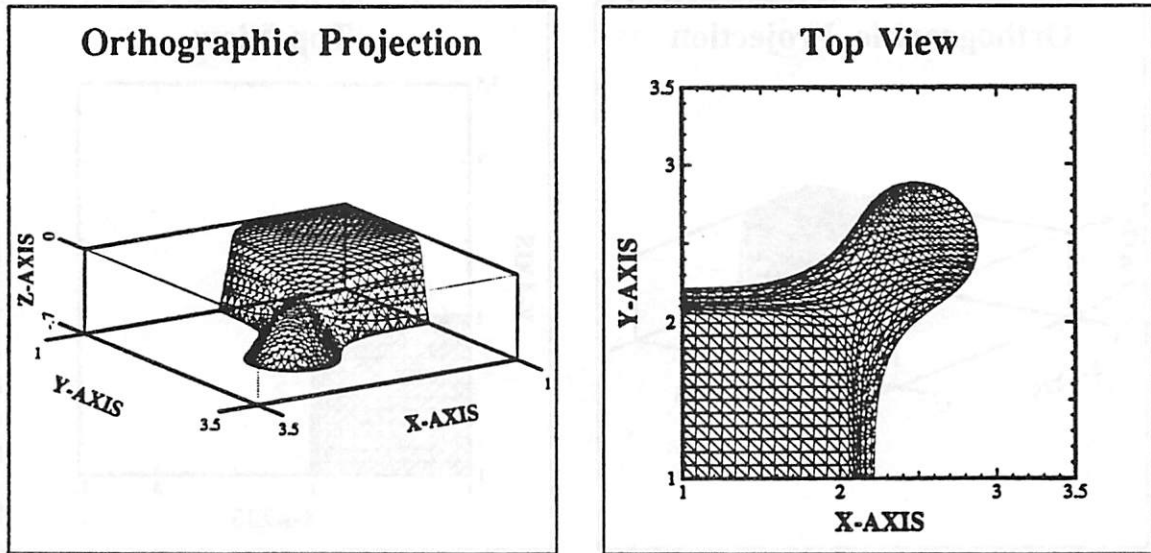
2D and 3D resist simulations have been run in an attempt to understand the development behavior of this opaque defect mask configuration. Figure 9.12 shows the 3D developed profiles on Shipley 1470 resist at best exposure. In Figure 9.12b, the resist foot left by the defect has a mesa-like structure; it is being developed laterally as well as vertically. As in the centered defect case discussed previously, the lateral dissolution dominates here while the vertical dissolution is somewhat slow. After 30 seconds of development time, the lateral dissolution fronts have crossed leaving a sharp almost 90° corner at the edge of the large square (Fig 9.12 c,d). There is clearly a difference between the 3D simulated resist profile and the constant intensity contours. The presence of the defect has slowed down the lateral dissolution and as a result the foot of the resist material is stretched from the 30% to approximately the 40% intensity level.

Figure 9.13 shows the underexposed and overexposed profiles on the same Shipley 1470 resist. Note that at 20% underexposure, the corner defect has printed; the resist profile appears to be very similar to the SEM of Figure 9.10b. However, the resist peninsula formed by the defect is on the verge of being etched away by the converging lateral dissolution fronts from the surrounding high intensity areas. At higher exposure doses, the defect is swept away by the converging lateral fronts, resulting in a sharp corner at the edge of the large resist square.

It is also instructive to compare the 3D simulations to *SAMPLE*. Figure 9.14 compares the 3D profiles along the diagonal to the 2D profiles generated using *SAMPLE* with *SPLAT* outlines. The multiple lines seen on the left set of plots are caused by loops formed by the

## Shipley 1470 : Best Exposure (Dose=80mJ/cm<sup>2</sup>)

(a) Development Time = 10 seconds



(b) Development Time = 20 seconds

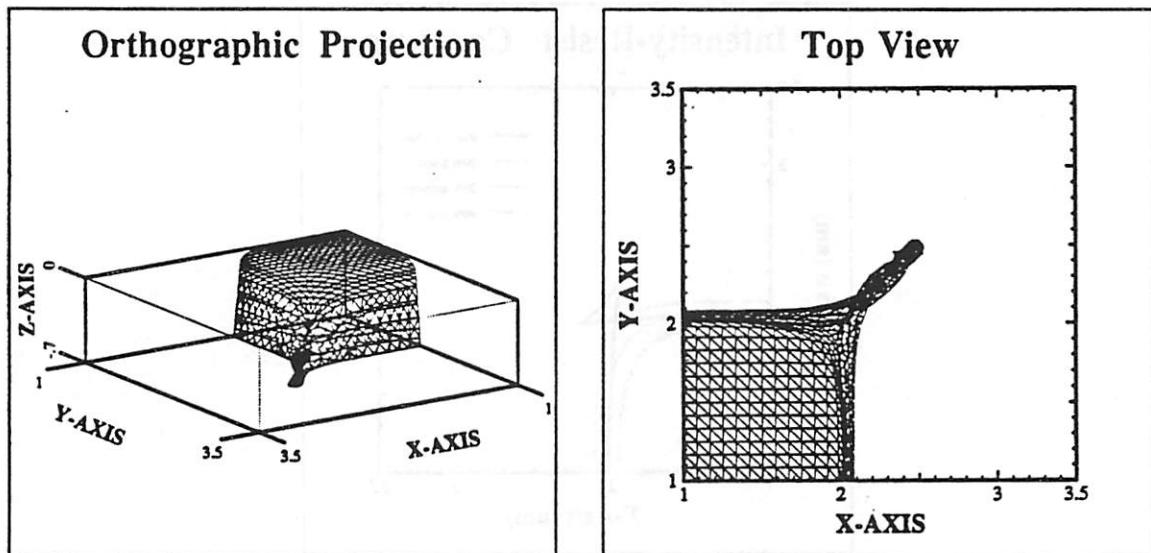
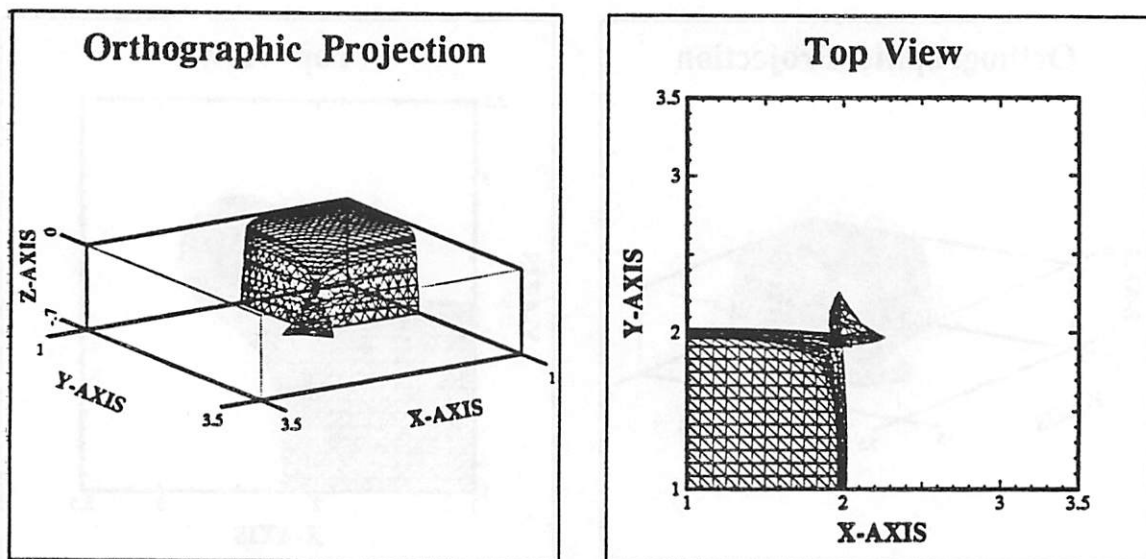


Figure 9.12 : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Shipley 1470 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

## Shipley 1470 : Best Exposure (Dose=80mJ/cm<sup>2</sup>)

(c) Development Time = 30 seconds



(d) Resist Profile vs Intensity Contours

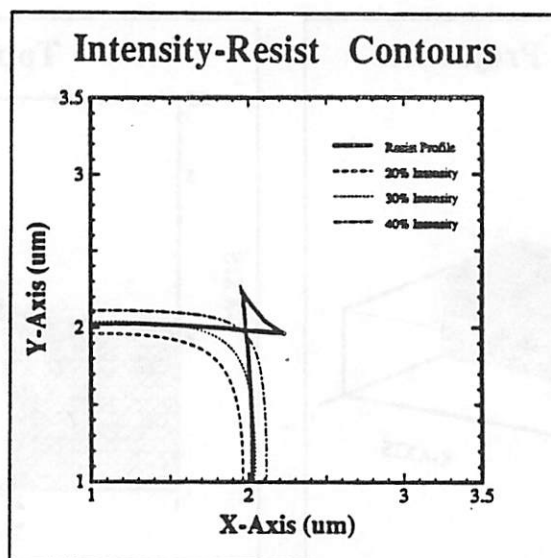
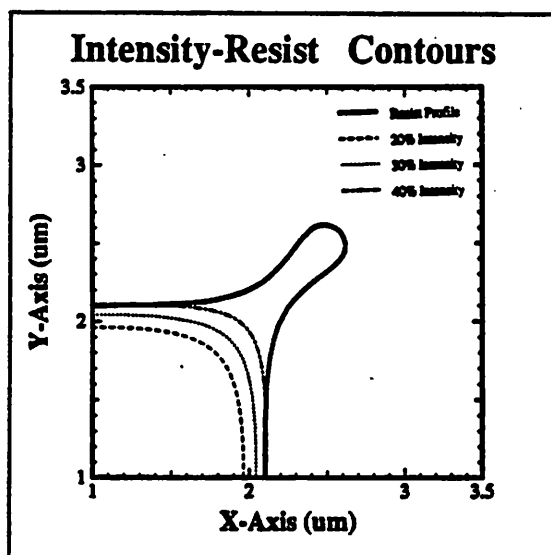


Figure 9.12 (cont) : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Shipley 1470 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

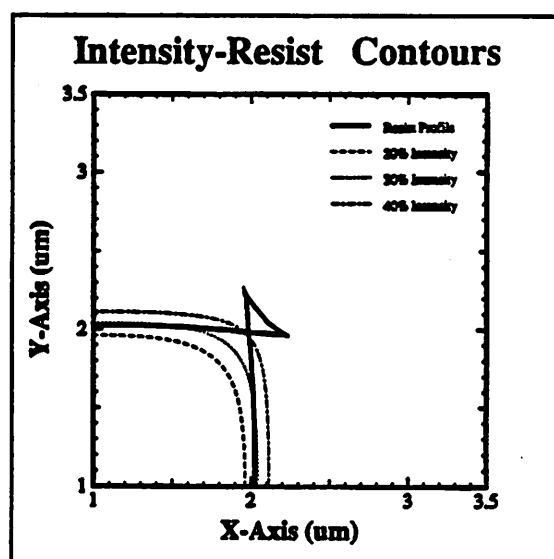


## Shipley 1470 : 30 sec Development Time

(a) Underexposed  
(Dose =  $64 \text{ mJ/cm}^2$ )



(b) Best Exposure  
(Dose =  $80 \text{ mJ/cm}^2$ )



(c) Overexposed  
(Dose =  $96 \text{ mJ/cm}^2$ )

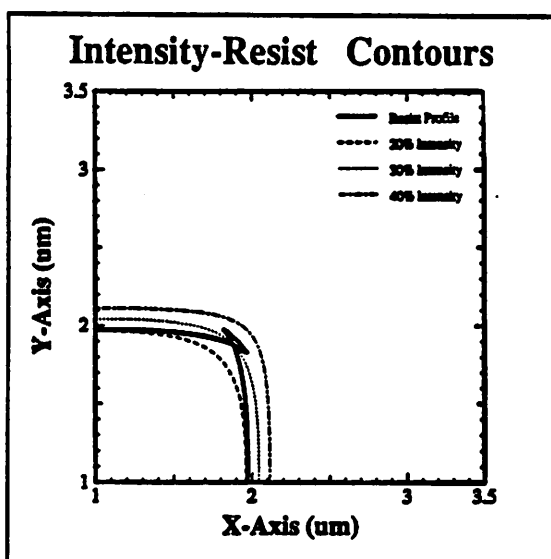


Figure 9.13 : 3D Resist Development simulated on Shipley 1470 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

## Shibley 1470 : 30 sec Development Time

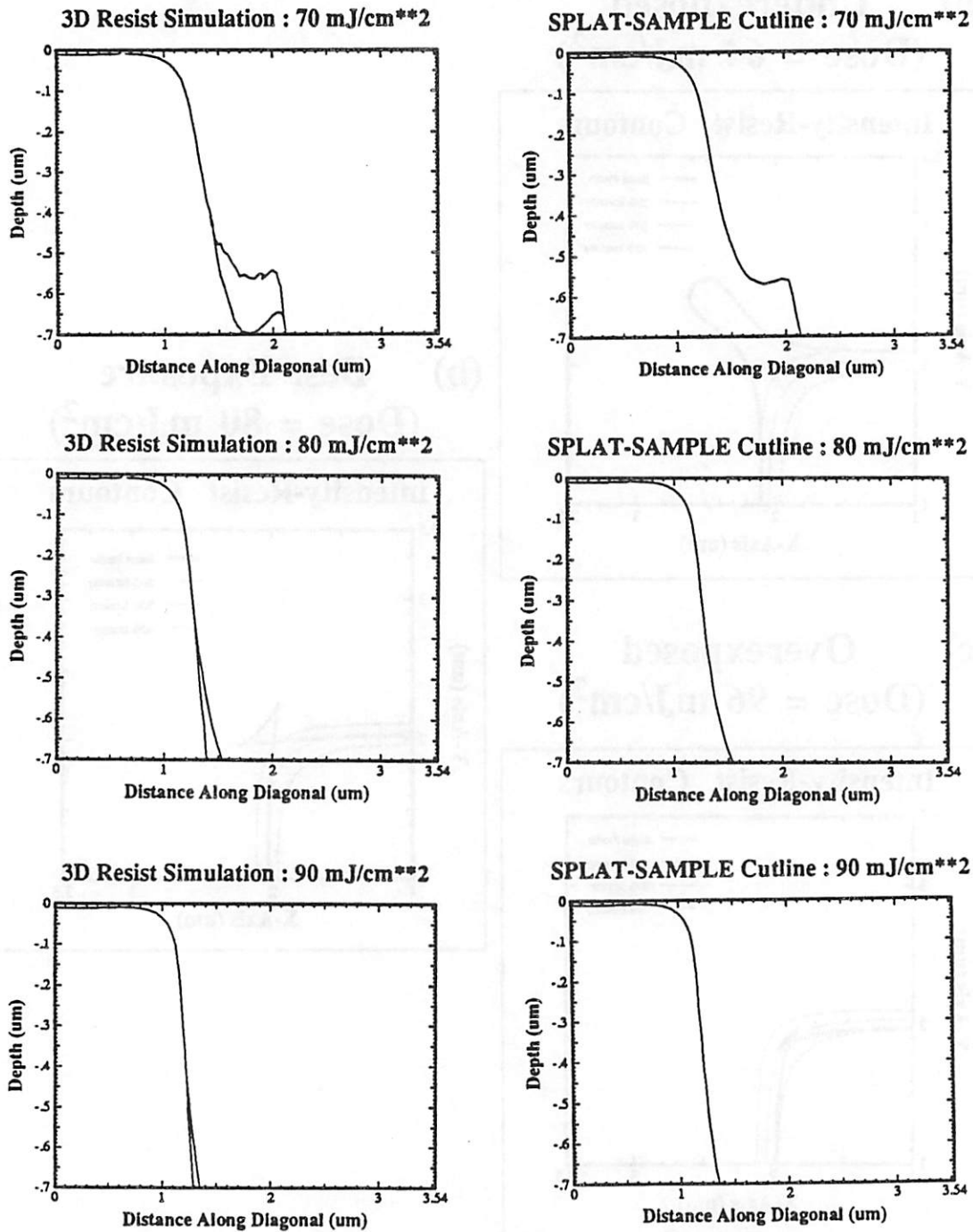


Figure 9.14 : Resist Development : 3D simulations vs 2D with cutlines.

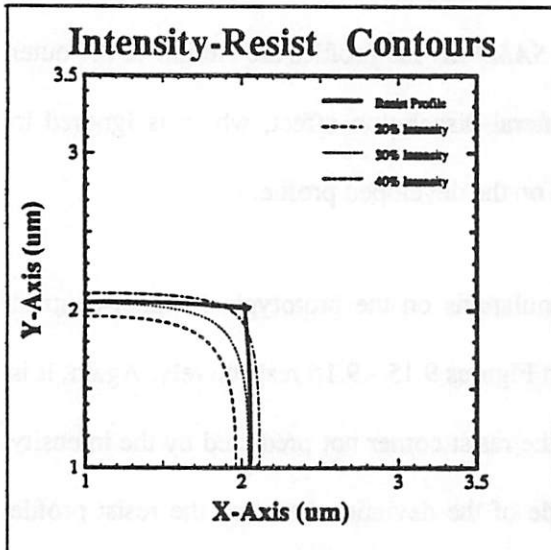
intersection of converging lateral dissolution fronts. The true developed profile is actually that of the innermost profile in each of the three plots on the right of Figure 9.14. In these plots, the outermost "looped" profile is due to the vertical (downwards) dissolution, while the innermost profile results from the intersection of converging lateral dissolution fronts. Comparing the two sets of curves, it is clear that the *SAMPLE 2D* profiles are similar to the outer "looped" curves of the 3D simulations. The lateral dissolution effect, which is ignored in *SAMPLE*, clearly has an dose-dependent impact on the developed profile.

A similar set of observations hold for simulations on the prototypical higher-contrast Olin Hunt resist. These simulations are shown in Figures 9.15 - 9.16 respectively. Again, it is seen that the defect has caused a sharpening of the resist corner not predicted by the intensity threshold model. And once more, the magnitude of the deviation between the resist profile and the constant intensity contours increases as the exposure dose is decreased. But the simulations also show that the lateral dissolution effect is not as pronounced as in the Shipley resist. There are no loops formed, and as can be seen in Figure 9.16, the 2D *SAMPLE* profiles are identical to those simulated using *SAMPLE-3D*.

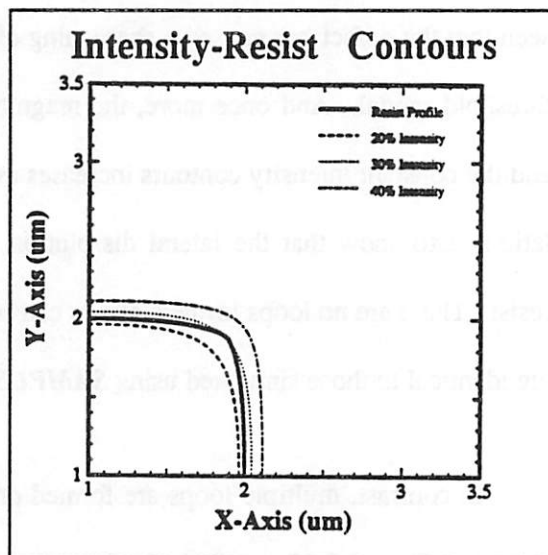
In contrast, multiple loops are formed on the surface-rate retarded Kodak 820 resist, as shown in Figures 9.17 and 9.18. Surface-rate retardation has increased the influence of lateral dissolution effects. As a result, the "true" developed profile can be determined only with a full 3D simulation. The 3D and 2D-cutline simulations have similar outer "looped curves, but the 3D simulations reveal that the true profile is very different from that predicted by the 2D-cutline simulations.

## Olin Hunt 6512 : 30 sec Development Time

- (a) Underexposed  
(Dose = 192 mJ/cm<sup>2</sup>)



- (b) Best Exposure  
(Dose = 240 mJ/cm<sup>2</sup>)



- (c) Overexposed  
(Dose = 288 mJ/cm<sup>2</sup>)

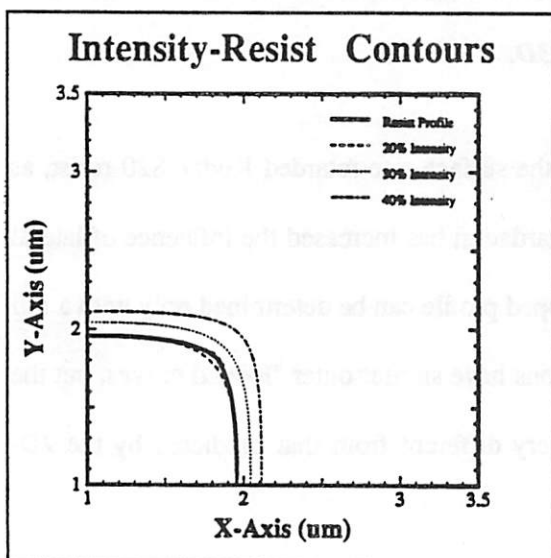


Figure 9.15 : 3D Resist Development simulated on Olin Hunt 6512 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

## Olin Hunt 6512 : 30 sec Development Time

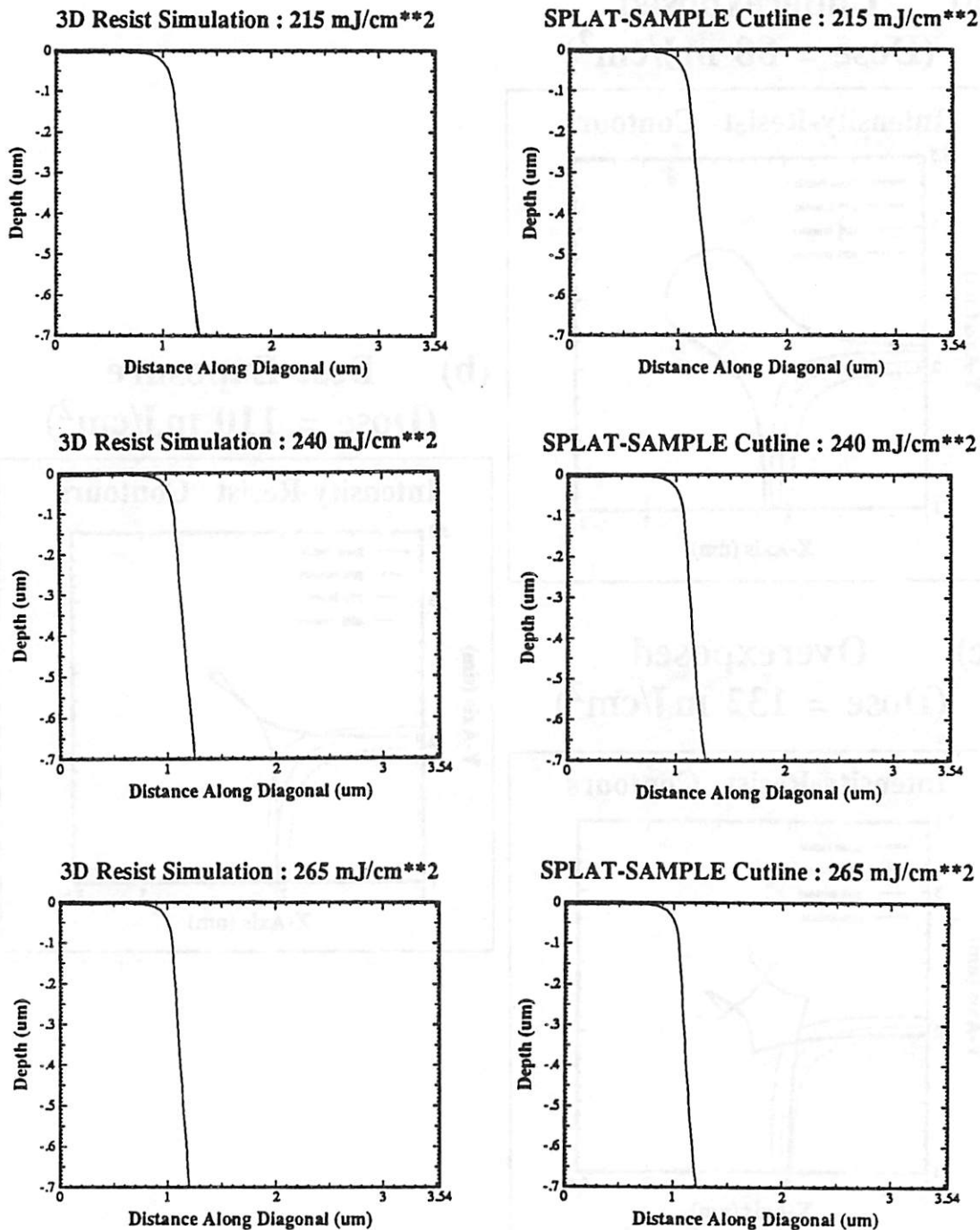
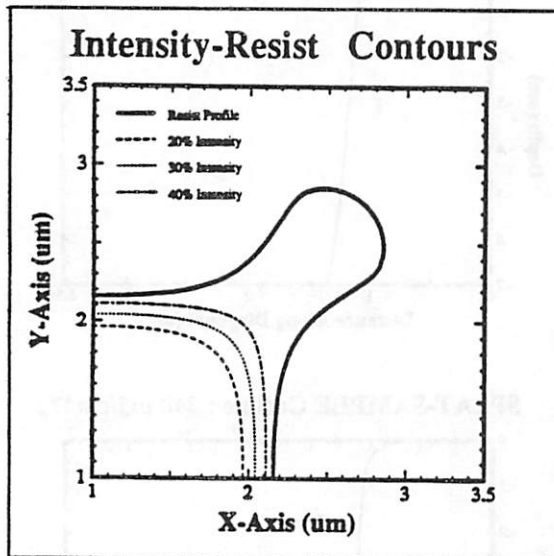


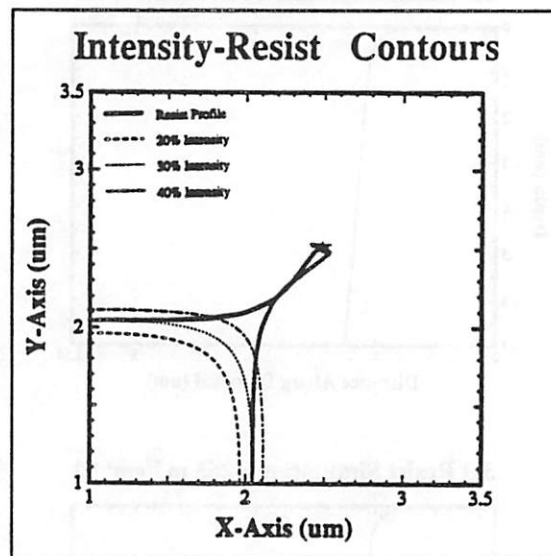
Figure 9.16 : Resist Development : 3D simulations vs 2D with cutlines.

## Kodak 820 : 30 sec Development Time

- (a) Underexposed  
(Dose =  $88 \text{ mJ/cm}^2$ )



- (b) Best Exposure  
(Dose =  $110 \text{ mJ/cm}^2$ )



- (c) Overexposed  
(Dose =  $132 \text{ mJ/cm}^2$ )

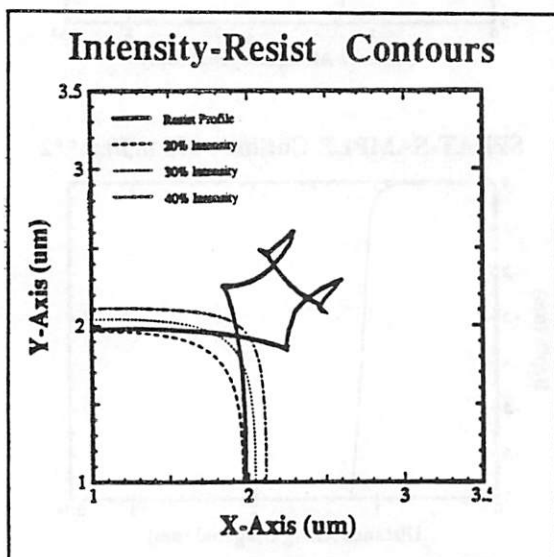


Figure 9.17 : 3D Resist Development simulated on Kodak 820 resist. Changes in the exposure dose affects the linewidth variation due to the defect.

## Kodak 820 : 30 sec Development Time

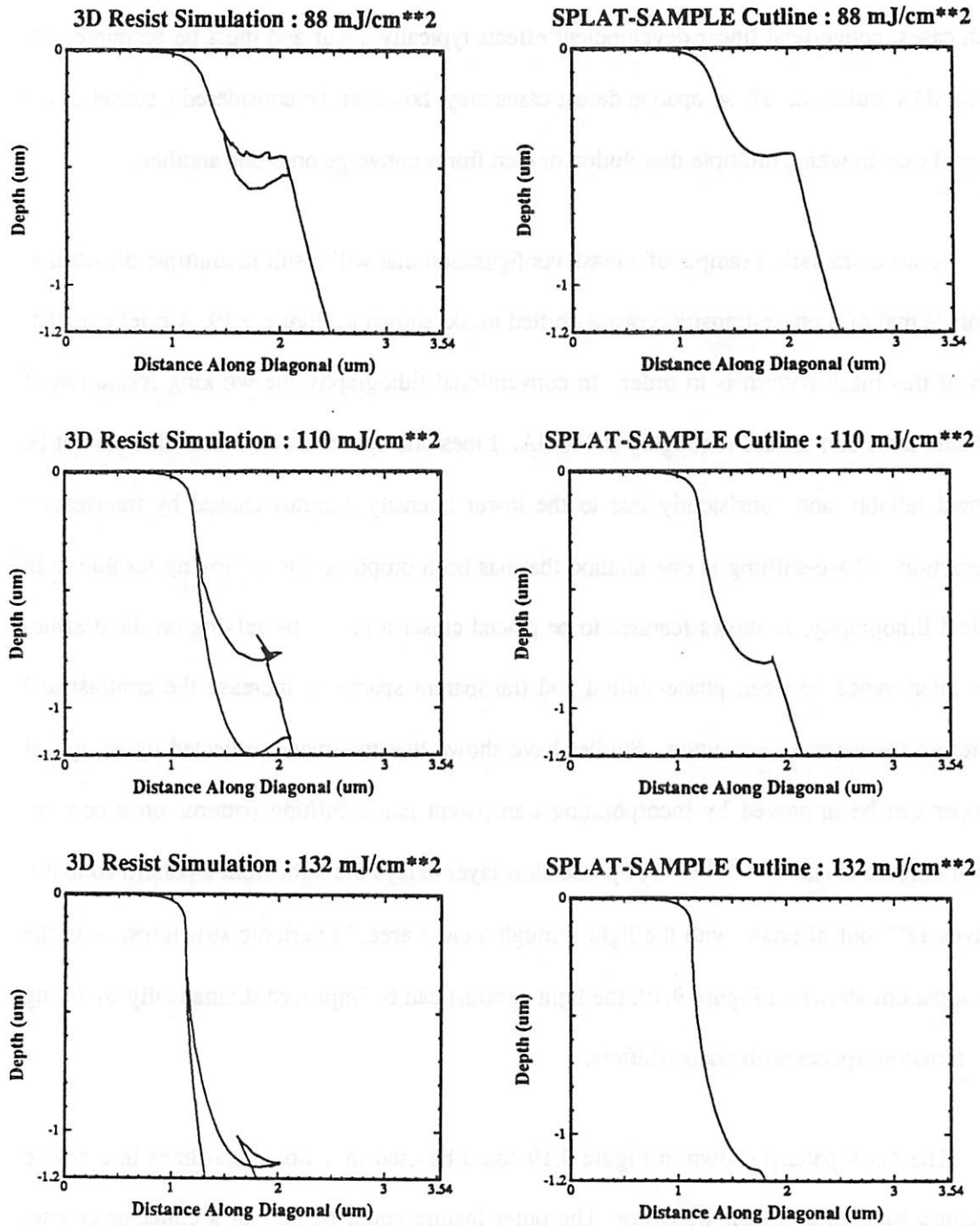


Figure 9.18 : Resist Development : 3D simulations vs 2D with cutlines.

#### 9.2.4. Adjoining Phase-Shifted Spaces

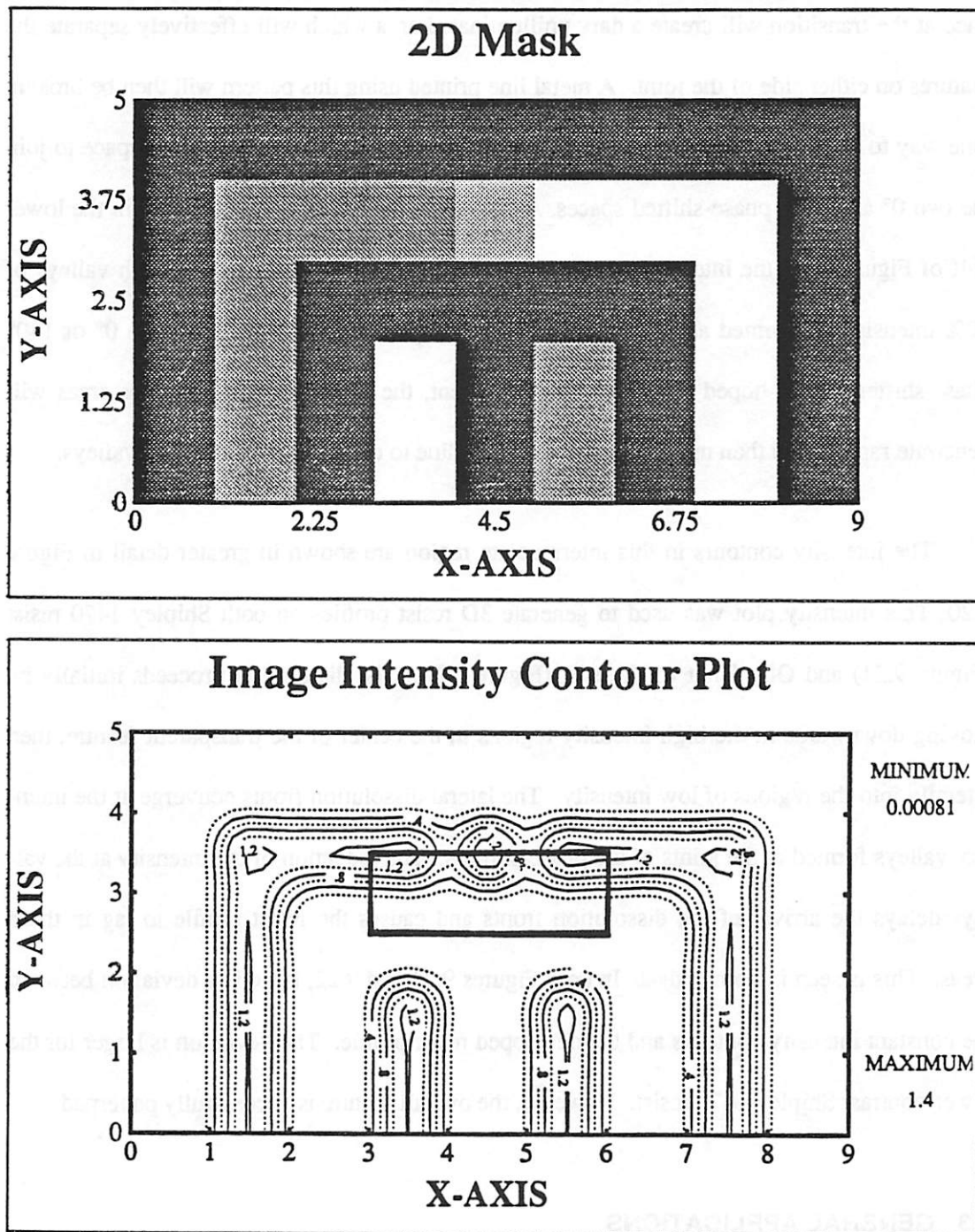
The two cases discussed previously both involved opaque defects in a clear area. In such cases, nonvertical linear development effects typically occur and must be accounted for using 3D simulation. These opaque defect cases may, however, be considered a subset of the general case in which multiple dissolution or etch fronts converge onto one another.

A more dramatic example of a mask configuration that will result in multiple dissolution fronts is that of a phase-transition phase-shifted mask, shown in Figure 9.19. A brief explanation of this mask pattern is in order. In conventional lithography the working resolution of periodic lines and spaces is roughly  $0.8 \lambda/NA$ . Lines and spaces smaller than this cannot be printed reliably and consistently due to the lower intensity contrast caused by interfeature interaction. Phase-shifting is one method that has been proposed for improving resolution in optical lithography. It allows features to be placed closer together by relying on the destructive interference between phase-shifted and transparent spaces to increase the contrast and therefore the working resolution. Studies have shown that the image projected by an optical stepper can be improved by incorporating transparent phase-shifting patterns on a conventional chrome mask.<sup>14, 15, 16, 17</sup> A phase-shift layer delays the light from a pattern so that it arrives  $180^\circ$  out of phase with the light through a clear area. In periodic structures, as in the mask pattern shown in Figure 9.19, the light contrast can be improved dramatically by filling in alternating spaces with phase-shifters.

The mask pattern shown in Figure 9.19 could be used to define metal lines in a device structure such as a bipolar transistor. The outer feature could be part of a collector contact which is to remain unbroken. In order to gain the most out of phase-shifting, the spaces and phase-shifters must be placed in alternate order. But there is a complication here in that there



## 1 $\mu\text{m}$ (0.64 $\lambda/\text{NA}$ ) Phase-Shifted Lines



**Figure 9.19:** Image intensity contour plot of a phase-shifted mask pattern. The simulation was run using  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA}=0.28$  and partial coherence  $\sigma = 0.5$ .

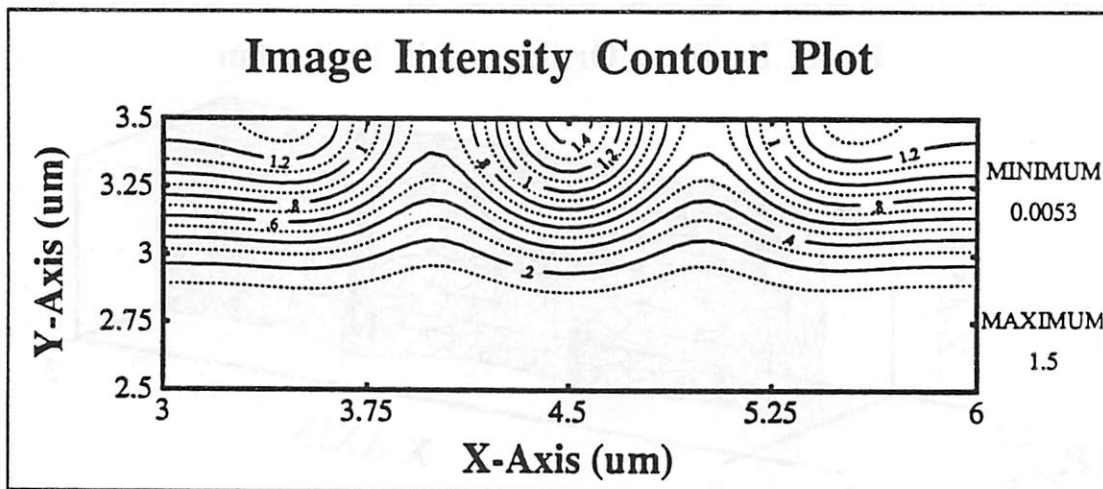
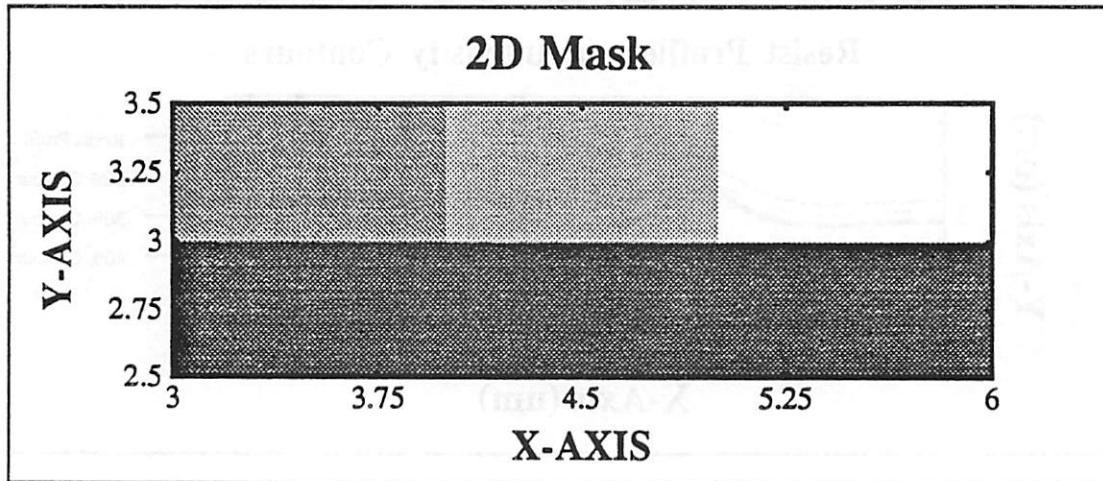
is a transition from a  $180^\circ$  phase-shifted outer elbow to a  $0^\circ$  phase-shifted elbow. But if a phase-shifted feature is joined abruptly to a non-phase-shifted feature, the destructive interference at the transition will create a dark unilluminated area which will effectively separate the features on either side of the joint. A metal line printed using this pattern will then be broken. One way to get around this problem is to use an intermediate  $90^\circ$  phase-shifted space to join the two  $0^\circ$  and  $180^\circ$  phase-shifted spaces. As shown in the intensity contour plot in the lower half of Figure 9.19, the intensity remains high along the joined features, although valleys of 70% intensity are formed at the areas where the  $90^\circ$  phase-shift meets either the  $0^\circ$  or  $180^\circ$  phase-shifters. It is hoped that during development, the dissolution in the peak areas will penetrate rapidly and then move laterally along the line to clear the 70% intensity valleys.

The intensity contours in this intermediate region are shown in greater detail in Figure 9.20. This intensity plot was used to generate 3D resist profiles on both Shipley 1470 resist (Figure 9.21) and Olin Hunt 6512 resist (Figure 9.22). The dissolution proceeds initially by moving downwards in the high intensity regions in the center of the transparent feature, then laterally into the regions of low intensity. The lateral dissolution fronts converge at the intensity valleys formed at the joints of the phase-shifters. The reduction of the intensity at the valleys delays the arrival of the dissolution fronts and causes the resist profile to lag in these areas. This is seen in both resists. In both Figures 9.21 and 9.22, there is a deviation between the constant intensity contours and the developed resist profile. The deviation is larger for the lower-contrast Shipley 1470 resist. However, the overall feature is successfully patterned.

### 9.3. GENERAL APPLICATIONS

*SAMPLE-3D* has been used to examine a number of interesting 2D mask patterns. This section will discuss briefly some applications of the 3D photolithography simulator.

## 1 $\mu\text{m}$ (0.64 $\lambda/\text{NA}$ ) Phase-Shifted Lines (Expanded Scale)



**Figure 9.20:** Image intensity contour plot of a phase-shifted mask pattern. The simulation was run using  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA}=0.28$  and partial coherence  $\sigma = 0.5$ .

## 1 $\mu\text{m}$ (0.64 $\lambda/\text{NA}$ ) Phase-Shifted Lines

Shipley 1470 : Dev. Time = 10 sec, Dose = 80  $\text{mJ}/\text{cm}^2$

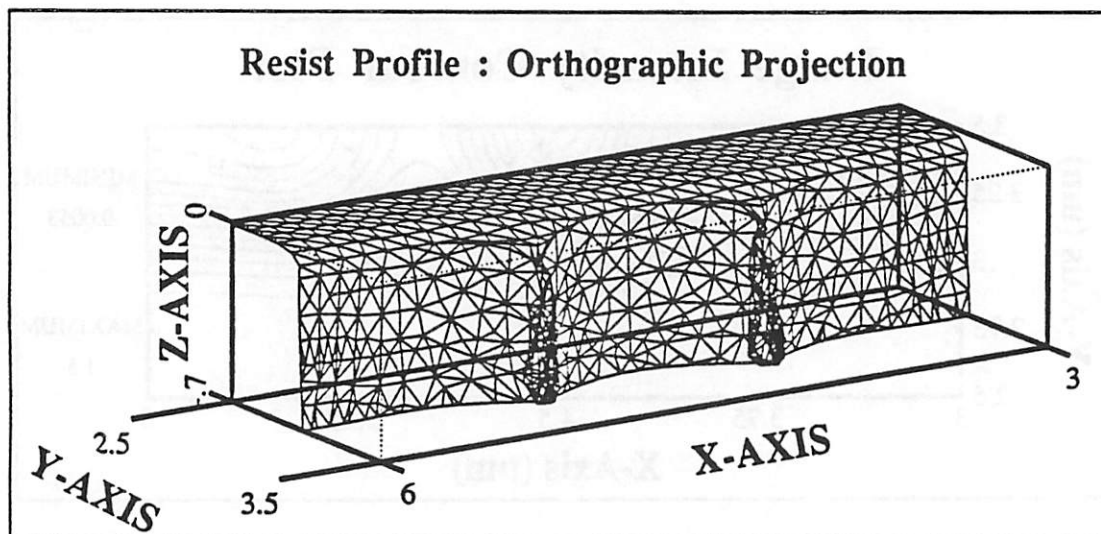
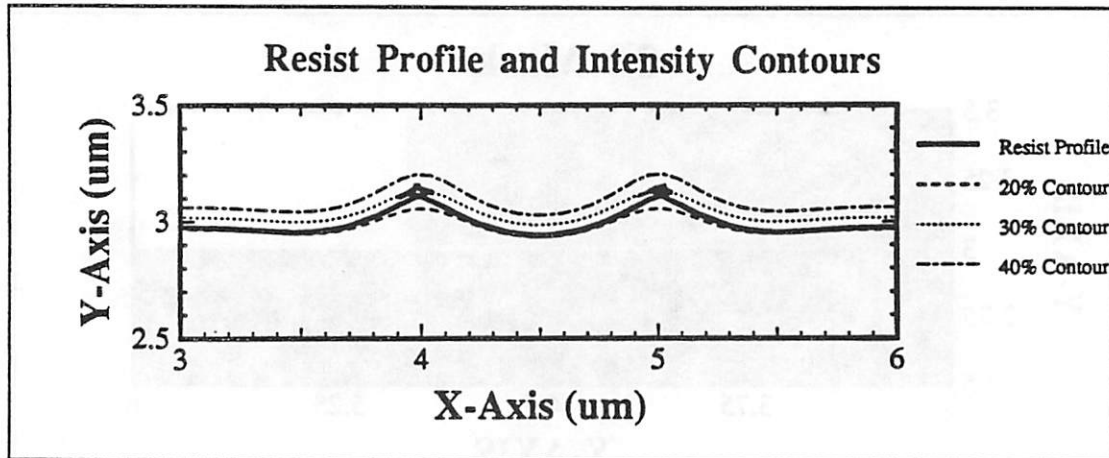


Figure 9.21 : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Shipley 1470 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

## 1 $\mu\text{m}$ (0.64 $\lambda/\text{NA}$ ) Phase-Shifted Lines

Olin Hunt 6512 : Dev. Time = 10 sec, Dose = 240  $\text{mJ}/\text{cm}^2$

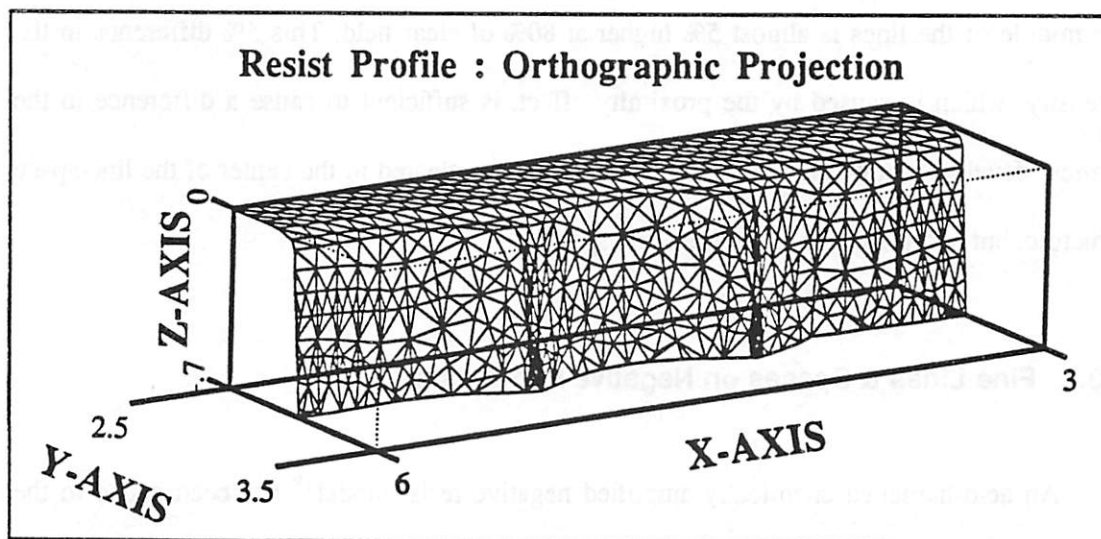
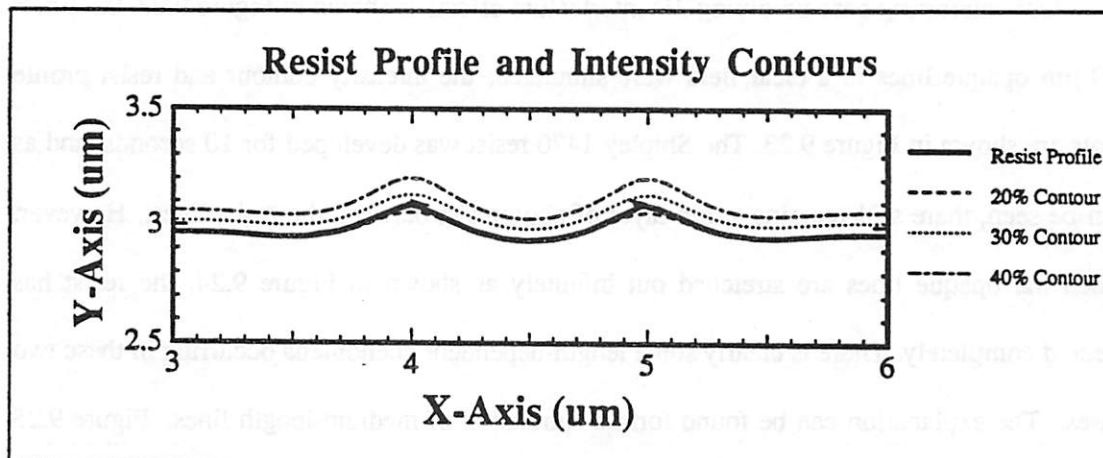


Figure 9.22 : 3D Resist Development, simulated on 0.7  $\mu\text{m}$  of Olin Hunt 6512 positive resist with 0.08  $\mu\text{m}$  of post-exposure bake diffusion.

Simulation results will be presented.

### 9.3.1. Clear Field Lines & Spaces

One interesting case involving 2D interfeature effects is shown in Figure 9.23.  $0.9\ \mu\text{m} \times 2.0\ \mu\text{m}$  opaque lines in a clear field were simulated; the intensity contour and resist profile plots are shown in Figure 9.23. The Shipley 1470 resist was developed for 10 seconds, and as can be seen, there still remains a thin layer of photoresist between the resist lines. However, when the opaque lines are stretched out infinitely as shown in Figure 9.24, the resist has cleared completely. There is clearly some length-dependent phenomena occurring in these two cases. The explanation can be found from a simulation of medium-length lines. Figure 9.25 shows the intensity contours and resist profiles of  $0.9\ \mu\text{m} \times 4\ \mu\text{m}$  lines. As can be seen from the intensity contour plot in Figure 9.25, there is an intensity saddle in between the tips of the opaque lines. The intensity here is approximately 75% of clear field, whereas the intensity in the middle of the lines is almost 5% higher at 80% of clear field. This 5% difference in the intensity, which is caused by the proximity effect, is sufficient to cause a difference in the vertical development of the photoresist. The space has cleared in the center of the line-space structure, but not at the tips of the lines and spaces.

### 9.3.2. Fine Lines & Spaces on Negative Photoresist

An acid-hardened chemically amplified negative resist model<sup>18</sup> has been added to the 3D resist-exposure simulator used in *SAMPLE-3D*. In Figure 9.26, simulation of a  $0.3\ \mu\text{m}$  equal line-space pattern is shown using the 3D photolithography simulator. Note that the width of the resist lines is narrower towards the substrate; this "necking" effect is observed experimentally.

## 0.9 $\mu\text{m}$ x 2 $\mu\text{m}$ Lines & Spaces in Clear Field

Shipley 1470 : Dev. Time = 10 sec, Dose = 80mJ/cm<sup>2</sup>

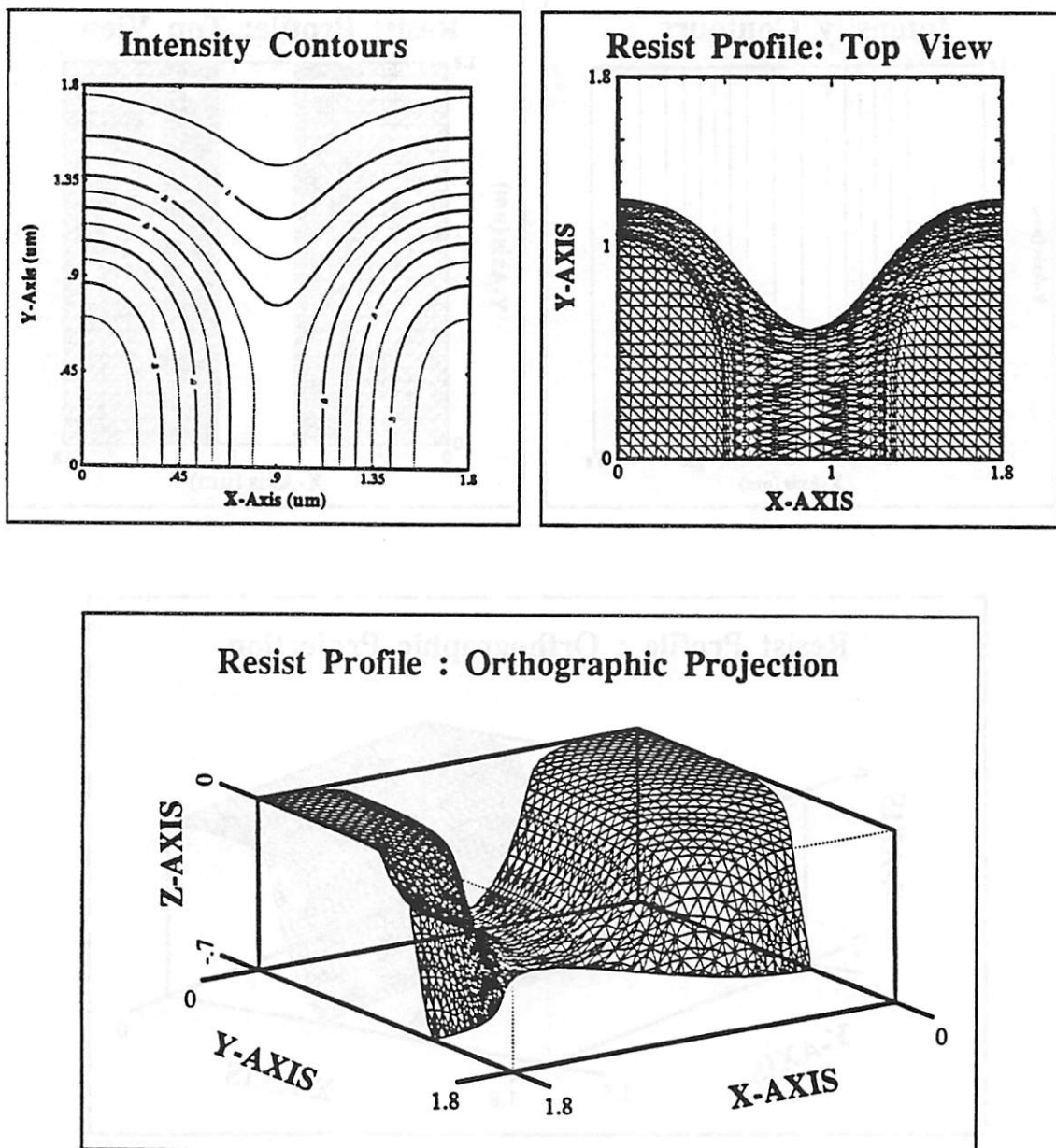


Figure 9.23 : 3D Intensity and Resist Development simulations.

## 0.9 $\mu\text{m}$ Infinite Lines & Spaces in Clear Field

Shipley 1470 : Dev. Time = 10 sec, Dose =  $80\text{mJ}/\text{cm}^2$

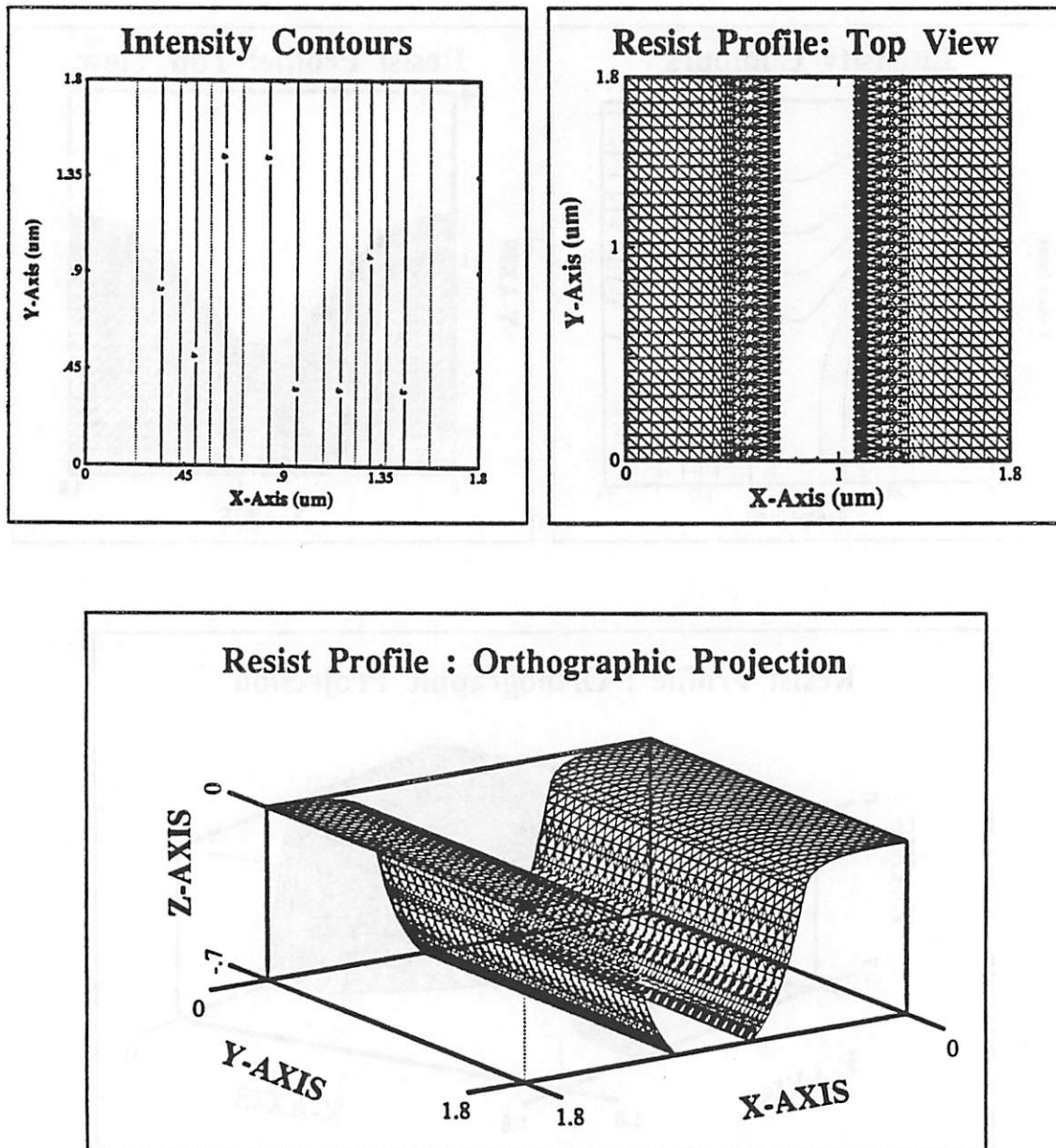


Figure 9.24 : 3D Intensity and Resist Development simulations.



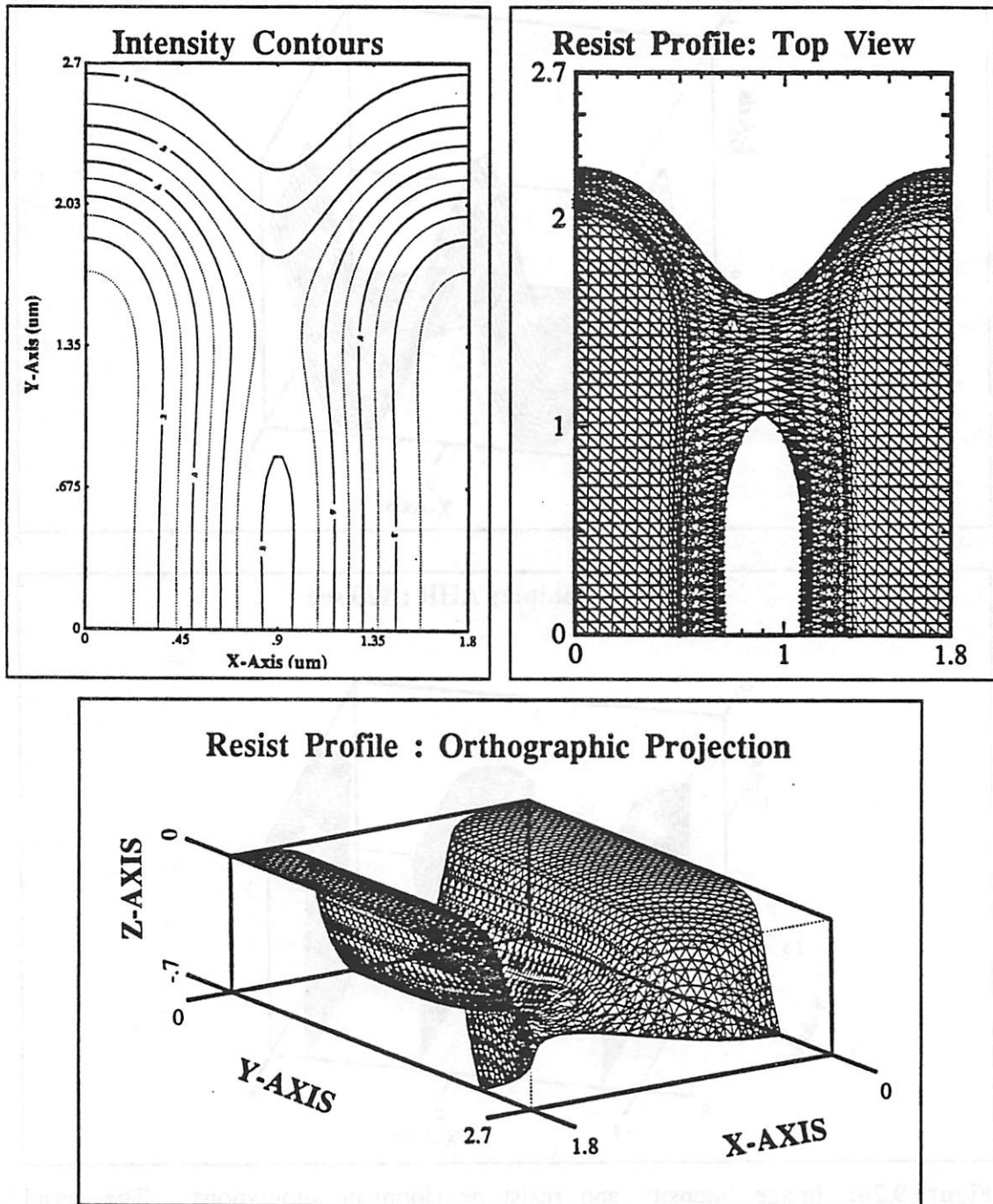
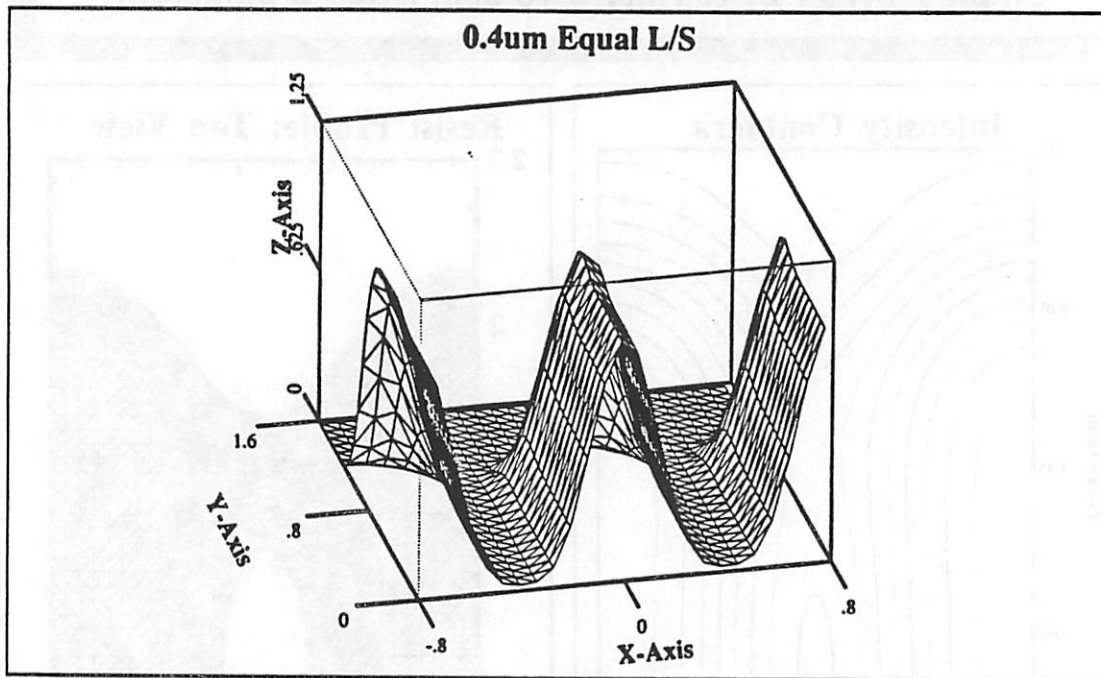
**0.9  $\mu\text{m}$  x 4  $\mu\text{m}$  Lines & Spaces in Clear Field****Shipley 1470 : Dev. Time = 10 sec, Dose = 80mJ/cm<sup>2</sup>**

Figure 9.25 : 3D Intensity and Resist Development simulations.

## 0.4 $\mu\text{m}$ Equally Spaced Lines



**Figure 9.26:** Image intensity and resist development simulations. The aerial image simulation was run using  $\lambda = 0.248 \mu\text{m}$ ,  $\text{NA}=0.42$  and partial coherence  $\sigma = 0.5$ . The dose was  $25.2 \text{ mJ/cm}^2$ , the bake was for  $140^\circ\text{C}$  for 60 seconds and the development time is 120 seconds.

### 9.3.3. Projection Lens Aberrations

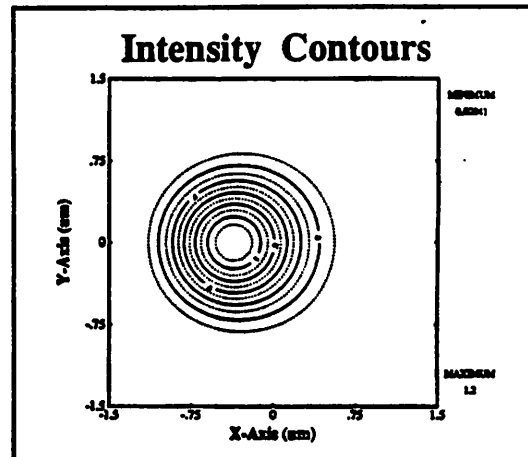
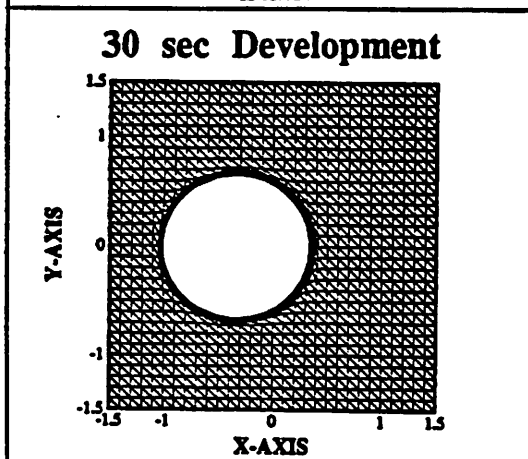
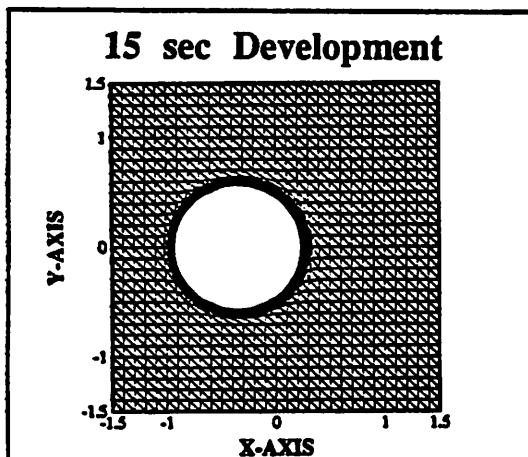
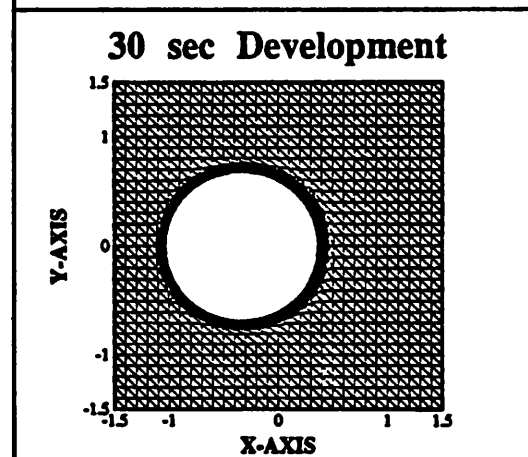
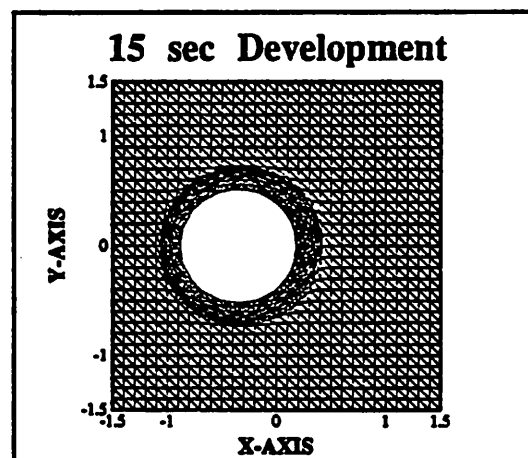
Coma and astigmatism are two primary lens aberrations which dominate image quality. The presence of aberrations is often recognized by the fact that contact holes are distorted from a shape with 90 degree rotational symmetry. The contour plots of the image intensity of an isolated transparent square under the influence of astigmatism and coma are shown in Figures 9.27 and 9.28 respectively. Also shown in these two figures are top views of the photoresist profiles printed on Shipley 1470 and Olin Hunt 6512 resist. The aerial image simulations were run with  $\lambda = 0.436\mu\text{m}$ ,  $\text{NA} = 0.28$ , and partial coherence  $\sigma = 0.3$ . The lens aberration was assumed to have a maximum optical path distance (OPD) of  $0.4\lambda$ , while the contact was defined to be at the field coordinates (1,0) which corresponds to the right-hand edge of a circular lens field.

Coma tends to produce an "ice-cream cone" effect and movement along a radial line.<sup>7</sup> The small amount of coma in Figure 9.27 produces a build-up of the intensity on the side of the feature towards the center of the field. The intensity slope is higher towards the negative  $x$ -axis, which causes the contacts to print with steeper slopes in this direction too. The wall angles of the contact are steeper on the Shipley 1470 resist than on the Olin Hunt resist. This difference in the wall angles might be passed on to subsequent pattern transfer processes. But aside from this uniform difference in the slope of the wall angles, there seems to be little difference between the profiles printed on the two resists.

For astigmatism, there is no movement and spreading is produced both inward and outward along a radial line. This results in a double ended "football" shape in the radial direction as shown in Figure 9.28. The printed resist profiles appear at first glance to be more circular than the intensity contours. But upon closer inspection, it is found that the 30% intensity

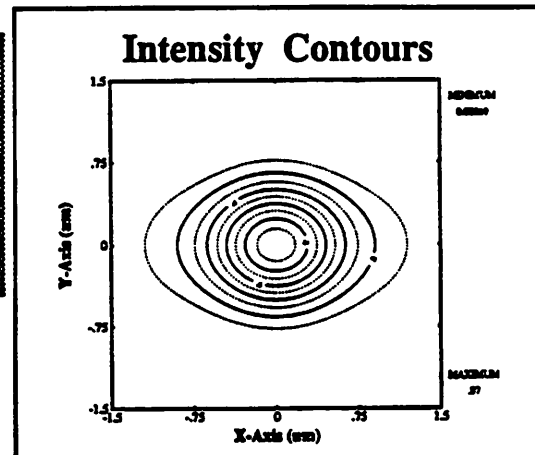
**Transparent Contact** **$0.8 \lambda/\text{NA} \times 0.8 \lambda/\text{NA}$** 

**$\sigma = 0.3$**

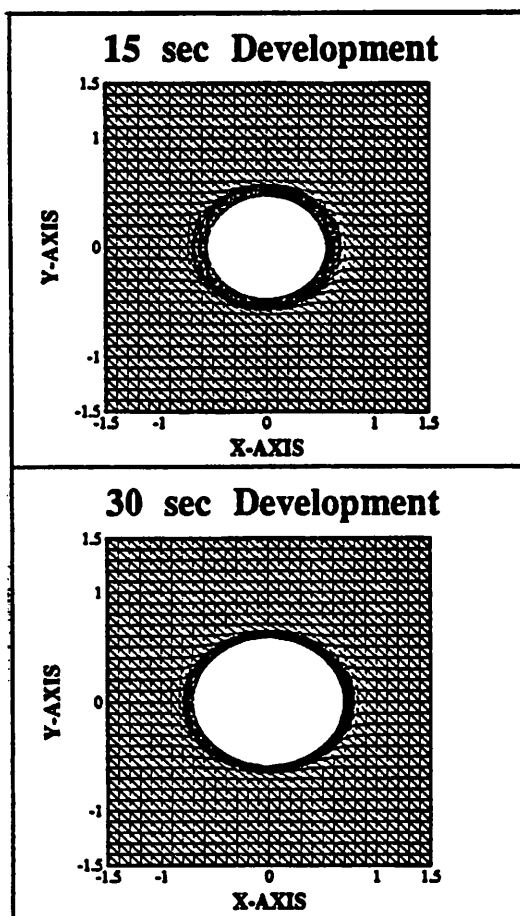
**Coma =  $0.4\lambda(1,0)$** **Shibley 1470****Dose =  $80 \text{ mJ/cm}^2$** **Olin Hunt 6512****Dose =  $240 \text{ mJ/cm}^2$** 

**Figure 9.27 : 3D Intensity and Resist Development simulations.**

**Transparent Contact**  
 **$0.8 \lambda/NA \times 0.8 \lambda/NA$**   
 **$\sigma = 0.3$**   
**Astigmatism =  $0.4\lambda(1,0)$**



**Shipley 1470**  
**Dose =  $80 \text{ mJ/cm}^2$**



**Olin Hunt 6512**  
**Dose =  $240 \text{ mJ/cm}^2$**

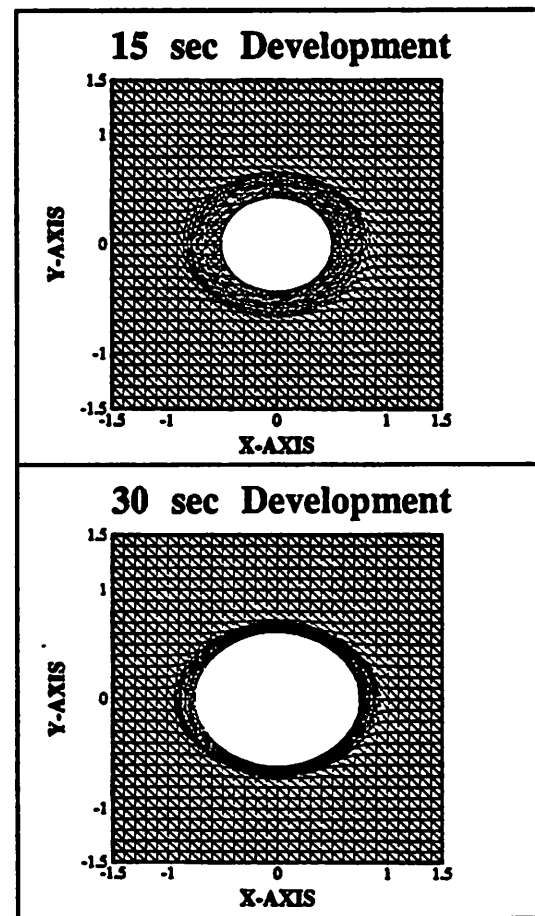


Figure 9.28 : 3D Intensity and Resist Development simulations.

contour agrees with the 3D second developed profile at the resist substrate intersection. The slopes of the resist profiles also reflect the change in intensity slope; the wall angles are steeper along the horizontal than along the vertical axis.

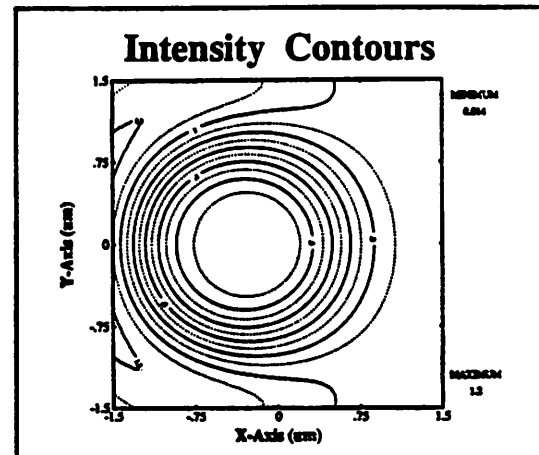
There are no surprises in the resist profiles of opaque contacts either. Figure 9.29 and 9.30 show the 3D resist profiles of a  $0.8 \lambda/NA$  square opaque defect simulated with coma and astigmatism respectively. Again, the resist profiles agree with the intensity threshold model. Astigmatism has produced a rectangular-shaped resist pad, while coma has left a circular but asymmetrical resist island.

#### 9.3.4. Isolated Contacts with Optimally-Placed Phase-Shifters

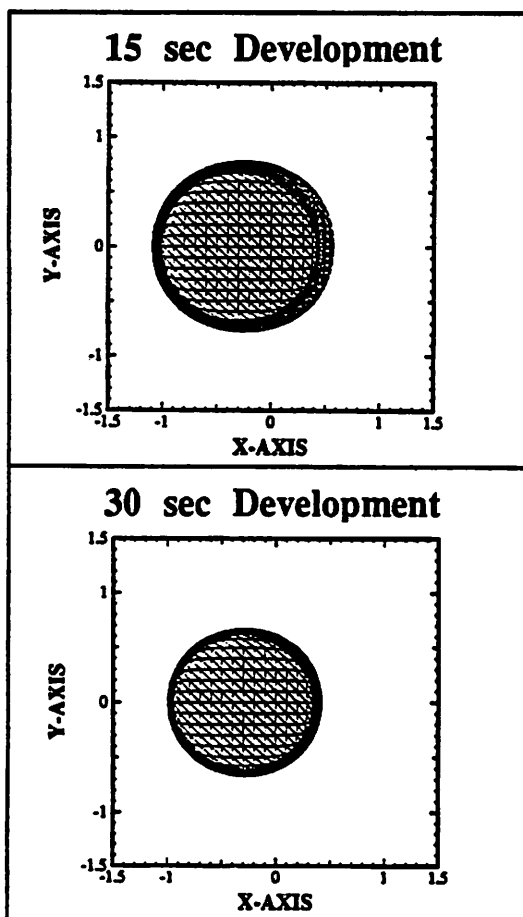
Small subimageable phase-shifters surrounding a transparent isolated contact can be used to increase the intensity of the light passing through the contact. Figure 9.31 compares the two-dimensional intensity profile and the three-dimensional developed resist profile of a  $0.9 \mu\text{m}$  ( $0.6 \lambda/NA$ ) square contact against that of the same contact surrounded by four  $0.3 \mu\text{m}$  ( $0.2 \lambda/NA$ ) phase-shifters. The phase-shifters are placed so that their centers are  $0.7 \lambda/NA$  from the center of the contact; it has been shown<sup>12</sup> that the  $0.7 \lambda/NA$  center-to-center distance optimizes the peak image intensity and image slope of the contact. The peak intensity of the phase-shifted contact is clearly greater than that of the conventional contact. The intensity profile of the phase-shifted contact also shows that a circular ring of low intensity light has formed around the main peak of the image. This ring is formed by the interaction between the phase-shifters and the secondary lobe of the contact.

The 3D simulations of the resist profile in Figure 9.31 were run on  $0.7133 \mu\text{m}$  of Olin Hunt 6512 resist, without and with post-exposure bake diffusion. Without any post-exposure

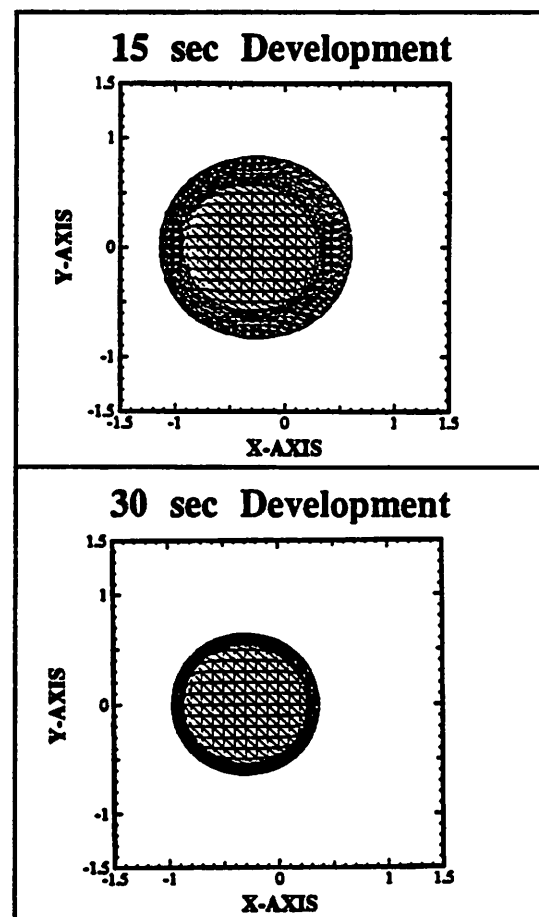
**Opaque Contact**  
 $0.8 \lambda/NA \times 0.8 \lambda/NA$   
 $\sigma = 0.3$   
**Coma =  $0.4\lambda(1,0)$**



**Shipley 1470**  
**Dose =  $80 \text{ mJ/cm}^2$**

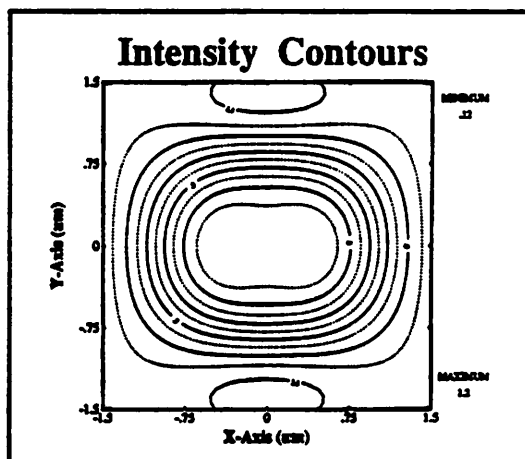


**Olin Hunt 6512**  
**Dose =  $240 \text{ mJ/cm}^2$**

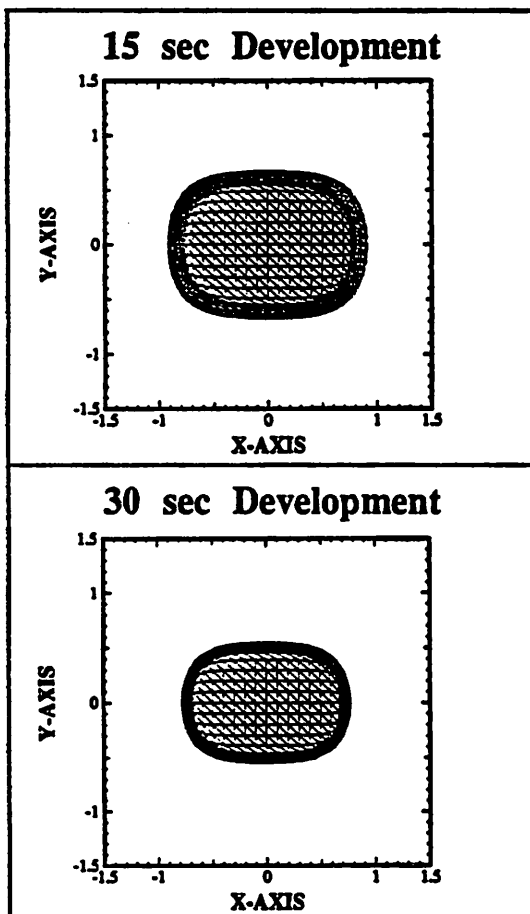


**Figure 9.29 : 3D Intensity and Resist Development simulations.**

**Opaque Contact**  
 **$0.8 \lambda/NA \times 0.8 \lambda/NA$**   
 **$\sigma = 0.3$**   
**Astigmatism =  $0.4\lambda(1,0)$**



**Shipley 1470**  
**Dose =  $80 \text{ mJ/cm}^2$**



**Olin Hunt 6512**  
**Dose =  $240 \text{ mJ/cm}^2$**

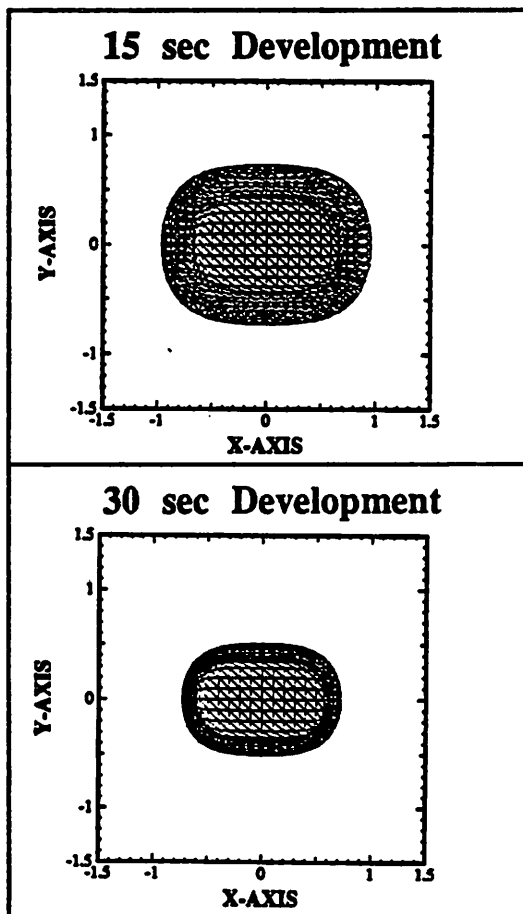
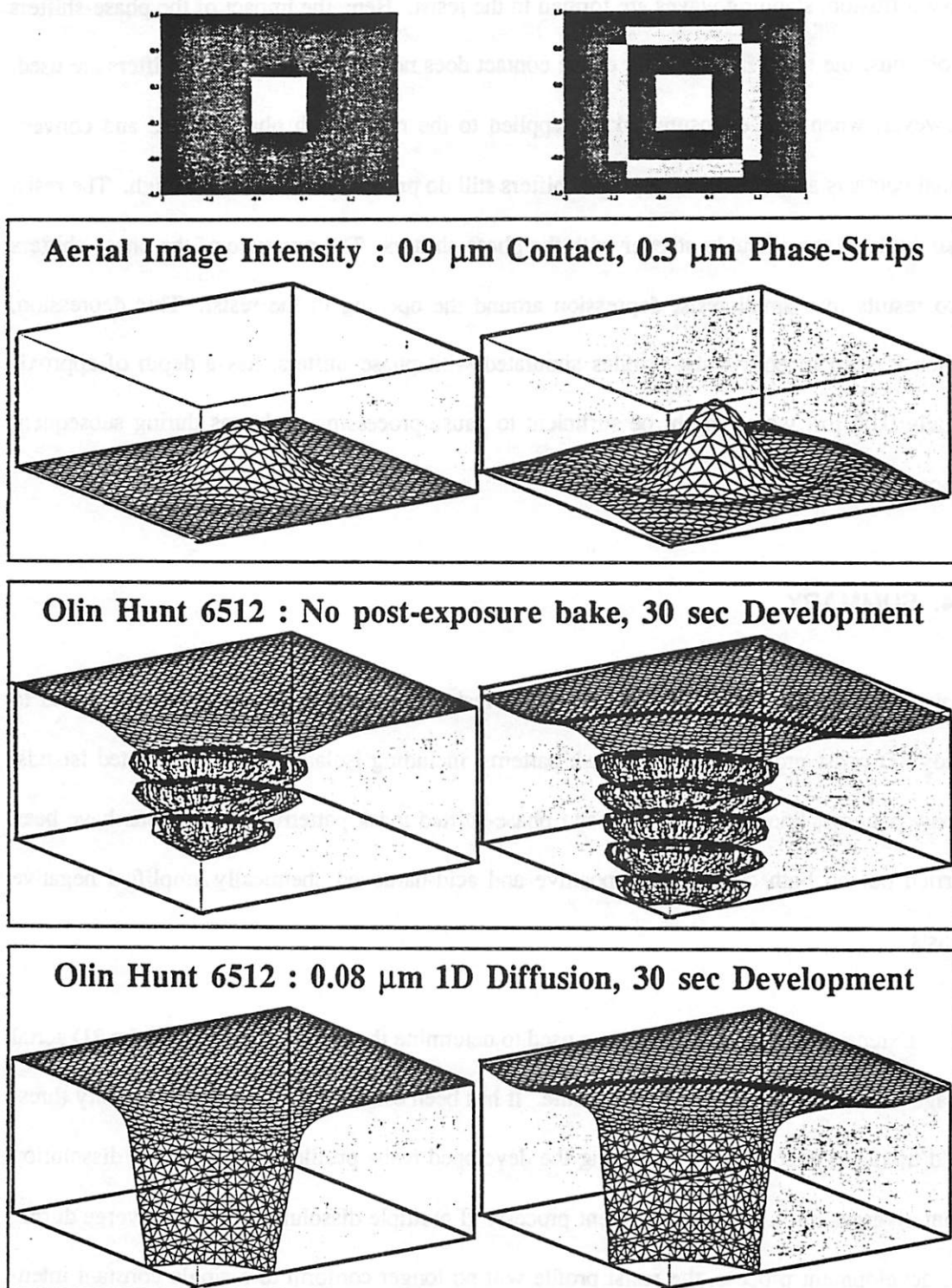


Figure 9.30 : 3D Intensity and Resist Development simulations.





**Figure 9.31 :** 2D aerial image and 3D resist development simulation of a contact with and without outrigger phase-shifters. The simulation used  $\lambda = 0.436 \mu\text{m}$ ,  $\text{NA} = 0.28$ ,  $\sigma = 0.5$ , resist thickness =  $0.713 \mu\text{m}$ .

bake diffusion, standing waves are formed in the resist. Here, the impact of the phase-shifters is obvious; the resist in the center of the contact does not clear unless phase-shifters are used. However, when post-exposure bake is applied to the resist, both phase-shifted and conventional contacts are opened. The phase-shifters still do provide some benefit though. The resist wall angle is considerably steeper with the phase-shifters. The presence of the phase-shifters also results in a small resist depression around the opening in the resist. This depression, which is seen in both resist profiles simulated with phase-shifters, has a depth of approximately 0.06  $\mu\text{m}$  which might be sufficient to cause processing problems during subsequent process steps.

#### **9.4. SUMMARY**

A working version of a 2D exposure and 3D resist development simulator has been used to study 3D resist profiles from 2D mask patterns, including isolated contacts, isolated islands, elbow patterns, line-space patterns, and phase-shifted mask patterns. Simulations have been carried out on both conventional positive and acid-hardened chemically amplified negative resists.

Extensive simulations have been used to determine the correlation between the 2D aerial image and the 3D developed resist profile. It has been established that the 2D intensity threshold model is adequate for predicting the developed resist profiles as long as the dissolution front diverges during the development process. If multiple dissolution fronts converge during the development process, the resist profile will no longer conform to a single constant intensity contour. In such situations, full 3D resist simulation is required to determine the resist profiles and thus to understand the printing process.

Opaque defects in a clear area provide the most vivid illustrations of the need for 3D resist development simulation. The lowering of the intensity in the vicinity of the defect slows down the vertical dissolution and allows lateral dissolution effects to come into play. Lateral dissolution will speed up the development in the neighborhood of the defect, and cause the formation of sharp protrusions in the resist material. The nonvertical dissolution effects predicted by 3D resist simulations appear to be in accordance to experimental observations. Simulations have also shown that the nonvertical resist development effects decrease as the exposure dose is increased. In addition, comparison of the intensity contours to the resist profile shows that the linewidth variations caused by the presence of the opaque defects are underestimated by the intensity threshold model.

## REFERENCES

1. P.D. Flanner III, S. Subramanian, A.R. Neureuther, "Two-Dimensional Optical Proximity Effects," *Proceedings of SPIE : Optical Microlithography V*, vol. 633, pp. 239-244, March 1986.
2. A.R. Neureuther, P. Flanner III, S. Shen, "Coherence of Defect Interactions with Features in Optical Imaging," *J. Vac. Sci. Technol. B.*, pp. 308-312, Jan/Feb 1988.
3. V. Mastromarco, A.R. Neureuther, K.K.H. Toh, "Printability of Defects in Optical Lithography : Polarity and Critical Location Effects," *J. Vac. Sci. Technol. B 6(1)*, pp. 224-229, Jan/Feb 1988.
4. B. Huynh, K.K.H. Toh, W.E. Haller, A.R. Neureuther, "Optical Printability of Defects in Two-Dimensional Patterns," *J. Vac. Sci. Technol. B 6(6)*, pp. 2207-2212, Nov/Dec 1988.
5. K.K.H. Toh, C.C. Fu, K.L. Zollinger, A.R. Neureuther, R.F.W. Pease, "Characterization of Voting Suppression of Optical Defects Through Simulation," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 194-202, March 1988.
6. K.K.H. Toh and A.R. Neureuther, "Mapping Image Quality in Optical Lithography with Visual Test Patterns," *Interface'89 : Proceedings of KTI Microelectronics Seminar*, pp. 155-177, November 1989.
7. K.K.H. Toh, A. R. Neureuther, "Identifying and Monitoring Effects of Lens Aberrations in Projection Printing," *Proceedings of SPIE : Optical Microlithography VI*, vol. 772, pp. 202-209, March 1987.
8. A.R. Neureuther, W.G. Oldham, B. Huynh, K. Toh, R. Ferguson, W.E. Haller, D. Sutija, "Test Patterns and Simulation Software for Characterization of Optical Projection Print-

- ing," *Interface'88 : Proceedings of KTI Microelectronics Seminar*, pp. 113-122, November 1988.
9. A.R. Neureuther, K.K.H. Toh, J.E. Fleishman, D. Yu, G. Misium, B. Huynh, B. Uathavikul, W.G. Oldham, "Exploratory Test Structures for Image Evaluation in Optical Projection Printing," *Proceedings of SPIE : Optical/Laser Microlithography II*, vol. 1088, pp. 83-91, March 1989.
  10. K.K.H. Toh, "Optical Lithography with Chromeless Phase-Shifted Masks," *Intel Internal Memorandum*, August 1990.
  11. K.K.H. Toh, G. Dao, R. Singh, H. Gaw, *Chromeless Phase-Shifted Masks : A New Approach to Phase-Shifting Masks*. Presented at the BACUS 10th Annual Symposium on Microlithography, September 1990.
  12. K.K.H. Toh, "Design Methodology for Dark-Field Phase-Shifted Masks," *Intel Internal Memorandum*, August 1990.
  13. K. Kadota, T. Ito, H. Fukui, M. Nagao, A. Sugimoto, M. Nozaki, T. Kato, "Resist Pattern Analysis for Submicron Feature Size using 3-D Photolithography Simulator," *Proceedings of SPIE : Optical/Laser Microlithography II*, vol. 1088, pp. 94-105, 1989.
  14. M.D. Levenson, N.S. Viswanathan, R.A. Simpson, "Improving Resolution in Photolithography with a Phase-Shifting Mask," *IEEE Transactions on Electron Devices*, vol. ED-29, no. 12, pp. 1812-1846, December 1982.
  15. I. Hanyu, S. Asai, K. Kosemura, H. Ito, M. Nunokawa, M. Abe, "New Phase-Shifting Mask with Highly Transparent SiO<sup>2</sup> Phase Shifters," *Proceedings of SPIE : Optical/Laser Microlithography III*, vol. 1264, pp. 167-177, 1990.
  16. T. Terasawa, N. Hasegawa, T. Kurosaki, T. Tanaka, "0.3-micron Optical Lithography using a Phase-Shifting Mask," *Proceedings of SPIE : Optical/Laser Microlithography*

*II*, vol. 1088, pp. 25-33, 1989.

17. A. Nitayama, T. Sato, K. Hashimoto, F. Shigemitsu, M. Nakase, "New Phase-Shifting Mask with Self-Aligned Phase-Shifters for a Quarter Micron Photolithography," *Proceedings of IEDM*, pp. 57-60, 1989.
18. R. Ferguson, J.M. Hutchinson, C.A. Spence, A.R. Neureuther, "Modeling and Simulation of a Deep-UV Acid Hardening Resist," *Electron, Ion and Photon Beam Sci. and Technol.*, 1990.

## CHAPTER 10

### CONCLUSIONS AND FUTURE RESEARCH

#### 10.1. Conclusions

The research described in this document has been aimed at developing a fast, robust and accurate computer program for the three-dimensional simulation of photoresist development. To achieve this goal, an algorithm for 3D development-etching was selected by examining a number of 2D etching algorithms, with emphasis paid to understanding and determining the conditions under which the algorithms would provide accurate results. A 3D development-etching simulator was then implemented and coupled to 2D imaging and 3D exposure simulators to form a complete 3D photoresist development simulator.

Photoresist development-etching algorithms using the cell, string and ray approaches have been implemented and utilized to examine basic tradeoffs in accuracy, delooping requirements, CPU time and memory requirements. The advantages and disadvantages of each algorithm are listed in Table 10.1. Cell-removal algorithms were found to be easy to implement but lacking in accuracy. These algorithms also require a considerable amount of memory, and are thus unsuitable for engineering workstation applications. The string method is fast and efficient, but is error-prone because incorrect surface representation excessively affects the calculation of the advancement vectors. Another key finding was that the ray algorithm produces incorrect results if the etch-rate changes too rapidly. Furthermore, the ray algorithm merely traces out selected points on the time-evolving surface; there may be ray-scarce regions that are not reached by the initial rays.

Comparison of 2D Etching Algorithms					
Method	Ease of Implementation	Memory Requirements	Computation Speed	Computation Time <sup>1</sup>	Comments
Cell (Ch.3)	Easy	Large	Slow	400 sec <sup>2</sup>	<i>Advantages</i> Algorithm is robust - no loops Boundaries, underlying topography handled easily <i>Disadvantages</i> Slow, Inefficient Inaccurate - faceting
Modified Cell (Ch.4)	Easy	Very Large	Very Slow	N/A <sup>3</sup>	<i>Advantages</i> Algorithm is robust - no loops Boundaries, underlying topography handled easily <i>Disadvantages</i> Slow, Inefficient Array requires lots of memory Inaccurate - discretization, faceting
String (Ch.6)	Difficult	Moderate	Fast	30 sec <sup>4</sup>	<i>Advantages</i> Fast, Efficient, Accurate <i>Disadvantages</i> Requires ordered mesh Needs boundary clipping, delooping Errors from incorrect surfaces - scissoring, rounding
Ray (Ch.6)	Easy	Fast	Small	N/A <sup>5</sup>	<i>Advantages</i> Fast, Efficient, Accurate Rays are independent of surface <i>Disadvantages</i> Needs boundary clipping Loops, Ray-Scarce Regions
Ray-String (Ch.7-8)	Moderate	Moderate	Moderate	80 sec <sup>6</sup>	<i>Advantages</i> Fast, Efficient, Accurate Nodes are independent of surface <i>Disadvantages</i> Needs boundary clipping Loops are formed

Table 10.1

1. Computation time measured on a SUN 4/280, for 2D simulation of 30-second resist development with the etch-rate distribution of Figure 2.3.
2. Cell simulation uses  $200 \times 100$  cells.
3. The 5-second development simulation lasts approximately 1 hour on the SUN4/280, and is inaccurate to boot.
4. The simulation begins with 50 segments, and uses a time-step of  $\Delta T = 0.092$  seconds.
5. The ray algorithm is unsuitable for resist simulation owing to the difficulty in reconstructing the surface from the rays.
6. The simulation begins with 50 segments, and uses a time-step of  $\Delta T = 0.1$  seconds. Recursive vector checking is used for accuracy. Simulation with  $\Delta T = 0.1$  seconds and no recursive vector checking lasts 67 seconds. Simulation with  $\Delta T = 0.5$  and recursive vector checking lasts 52 seconds.



An approach in which etch-rate-dependent rays are used to advance a string-like mesh has been found to be the most advantageous for the simulation of etching. This combined ray and string algorithm is computationally fast and accurate, insensitive to errors in the local etch-surface, and has modest memory requirements. The issues affecting the accuracy and efficiency of the ray-string simulation have been examined in some detail by implementing the algorithm in two dimensions. A recursive procedure that dramatically increases the accuracy of the ray-trajectory calculations has been developed and incorporated into the algorithm. The use of arc interpolation, small time-steps, small segment sizes and small maximum allowable segment lengths also increases the accuracy of the simulations, but at a cost of increased computation time.

The combined ray-string algorithm has been implemented in three dimensions in the C programming language. The data-structure uses multiple linked-lists to represent the nodes, segments and triangles that make up the etch boundary mesh. This hierarchical data-structure is well suited for implementing a number of mesh operations which are vital for preserving the balance between accuracy and efficiency. These operations include mesh modification, boundary clipping and delooping of the etch boundary surface. The capability for plotting and arbitrarily clipping the 3D surface has been added.

The 3D ray-string etch algorithm is the key element of a new fast, robust and accurate 3D photolithography simulator called *SAMPLE-3D*. This simulator integrates the 3D ray-string etch simulator with 2D imaging and 3D resist-exposure simulators, while also providing display and print capabilities. *SAMPLE-3D* has been used to study 3D resist profiles from 2D mask patterns, including isolated contacts, isolated islands, elbow patterns, line-space patterns as well as phase-shifted mask patterns. Simulations have been carried out on both medium-contrast and high-contrast positive and acid-hardened chemically amplified negative resists.

These simulations have provided valuable insight on the role of nonvertical resist dissolution effects and resist contrast on the pattern printing process.

## 10.2. Future Research

The *SAMPLE-3D* simulation program is a powerful tool for studying photolithography. *SAMPLE-3D* has been used to briefly examine a number of interesting issues in optical lithography. However, further systematic simulations are needed to characterize 3D dissolution effects, and to fully determine the impact of these effects on the pattern printing process. The simulation results should also be compared to experimental data in the near future.

*SAMPLE-3D* can also benefit from improvements and extensions to the 3D development-etching simulator. There are two primary areas for further research, which are :

### Delooping

The delooping of the mesh boundary surface must be made more reliable, more robust, and faster. At present, the deloop procedure uses a brute force approach in which all the triangles in the mesh are examined against each other for intersections. A plane-sweep algorithm could be implemented to decrease the number of intersection tests; this would decrease the computation time from  $O(N^2)$  to  $O(N\log N)$ ,  $N$  being the number of triangles in the mesh. In addition, delooping could benefit from a more intelligent approach in which only loop-prone areas are examined. One way in which this could be done is to rely on user-interaction, in which the user specifies a volume or a part of the surface to be delooped.

### Clipping Against Underlying Topography

In the future, the etch simulator should also be extended to handle underlying

topography. Presently, the simulation is carried out within a rectangular boundary, and the mesh is clipped against a constant  $z$ -plane. Addition procedures should be added to enable clipping against an arbitrary 3D surface representing the surface underlying the photoresist. An alternate approach, described in Chapter 7, is to drop the etch-rate to zero outside the boundary. However, in such an approach, care must be taken to gradually ramp down the etch-rate, so as to ensure that the rays do not bounce off any abrupt "reflective" boundaries.

*SAMPLE-3D* would also benefit from links to more accurate and rigorous 3D resist-exposure simulators such as *TEMPEST*<sup>1</sup> or M.Yeung's vector-based simulator.<sup>2</sup> Pattern transfer simulators could be added to *SAMPLE-3D*, for the simulation of additional processing steps. In addition, simulators for electron-beam or X-ray lithography could be used to generate etch-rate distribution data for use in the 3D etching simulator described in this document. The integration of these simulation tools can be done with relative ease since *SAMPLE-3D* is written in a C-shell script. A full simulator integrating a variety of 3D simulation tools would undoubtedly be of great use for understanding the many complex tradeoffs between materials, exposure tools and wafer conditions that govern the pattern transfer process.

## REFERENCES

1. J. Gamelin, R. Guerrieri, A.R. Neureuther, "Exploration of Scattering from Topography with Massively Parallel Computers," *J. Vac. Sci. Technol. B.*, 1989.
2. M.S. Yeung, "Modeling High Numerical Aperture Optical Lithography," *Proceedings of SPIE : Optical/Laser Microlithography*, vol. 922, pp. 149-167, March 1988.

**APPENDIX A**  
**DERIVING THE DIFFERENTIAL RAY EQUATION**

## APPENDIX A

### DERIVING THE DIFFERENTIAL RAY EQUATION

#### A.1. Solving the Least Action Problem with Variational Calculus

The problem of least action, as discussed in Chapter 5, is to find conditions that will minimize the expression

$$U = \int_C n(x, y, z) ds = \int_{P_i}^{P_f} n(x, y, z) ds \quad [\text{A.1}]$$

where  $n(x, y, z)$  is an arbitrary continuous function. The integral is evaluated over the curve  $C$ , and the limits of integration are fixed at points  $P_i$  and  $P_f$ . † If the variable  $U$  is to be a minimum, then to first order, the variation about  $U$  must be zero. In other words, to first order, there must be no change in the quantity  $U$  as the curve  $C$  over which the integral is evaluated is changed slightly.

$$\delta U = \delta \int_{P_i}^{P_f} n(x, y, z) ds = 0 \quad [\text{A.2}]$$

Applying the Chain Rule,

$$\delta U = \int_{P_i}^{P_f} \left[ \frac{\partial n}{\partial x} \delta x + \frac{\partial n}{\partial y} \delta y + \frac{\partial n}{\partial z} \delta z \right] ds = 0 \quad [\text{A.3}]$$

The problem now is to find an equation for the curve  $C$  such that the minimum equation above is satisfied.

The starting point is the relationship of the line element  $ds$  to its projections on a cartesian coordinate system. The line element  $ds$  is related to its projections  $dx$ ,  $dy$  and  $dz$  on the

---

† The derivation in this section follows substantially from the analysis of Carll,<sup>1</sup> who refers to it as a method of "decided advantages" for problems involving three coordinate axes. Carll's derivation is over only two dimensions  $x$  and  $z$ , but as shown in this section, it is easily extended to three dimensions.

axes of a rectangular system by Pythagoras' Theorem.

$$ds^2 = dx^2 + dy^2 + dz^2 \quad [A.4]$$

$$1 = \left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2 + \left(\frac{dz}{ds}\right)^2$$

$$1 = x'^2 + y'^2 + z'^2 \quad [A.5]$$

In order to simplify the equations, the ' notation is used to denote differentiation with respect to  $s$ , i.e.

$$x' = \frac{dx}{ds} \quad y' = \frac{dy}{ds} \quad z' = \frac{dz}{ds}$$

$$x'' = \frac{d^2x}{ds^2} \quad y'' = \frac{d^2y}{ds^2} \quad z'' = \frac{d^2z}{ds^2}$$

Now, if [A.5] is differentiated with respect to  $s$ , the following equation is obtained.

$$x'x'' + y'y'' + z'z'' = 0 \quad [A.6]$$

This can also be written in variational form as

$$x'\delta x' + y'\delta y' + z'\delta z' = 0 \quad [A.7]$$

where  $\delta x'$  is defined as the change in  $x'$  as the variable  $x$  is changed to  $x + \delta x$ .

$$\delta x' = \delta\left(\frac{dx}{ds}\right) = \frac{d}{ds}(\delta x)$$

Using this relationship, Equation [A.7] can be rewritten as

$$x' \frac{d}{ds}(\delta x) + y' \frac{d}{ds}(\delta y) + z' \frac{d}{ds}(\delta z) = 0 \quad [A.8]$$

Since [A.8] is true, any integral which has [A.8] as a factor must also be zero. Therefore, if

[A.8] is multiplied by  $n(x, y, z)$  and integrated over the curve  $C$ , the result will still be zero.

$$\int_{P_i}^{P_f} n(x, y, z) \left[ x' \frac{d}{ds}(\delta x) + y' \frac{d}{ds}(\delta y) + z' \frac{d}{ds}(\delta z) \right] ds = 0 \quad [A.9]$$

Now, in the definition of the problem, it was specified that the curve  $C$  over which the integration is being performed is fixed at the points  $P_i$  and  $P_f$ . So there is no variation with  $s$  at the end-points, and

$$x'(P_i) = x'(P_f) = y'(P_i) = y'(P_f) = z'(P_i) = z'(P_f) = 0$$

Integrating [A.9] by parts and using the end-point conditions specified above,

$$\int_{P_i}^{P_f} \left[ (nx')\delta x + (nz')\delta y + (nz')\delta z \right] ds = 0 \quad [\text{A.10}]$$

The whole point of the exercise so far is to obtain a zero-valued expression in terms of  $\delta x$ ,  $\delta y$  and  $\delta z$ . Now, when Equation [A.10] is subtracted from [A.3], the result is

$$\delta U = \int_{P_i}^{P_f} \left[ \left\{ \frac{\partial n}{\partial x} - (nx')' \right\} \delta x + \left\{ \frac{\partial n}{\partial y} - (ny')' \right\} \delta y + \left\{ \frac{\partial n}{\partial z} - (nz')' \right\} \delta z \right] ds \quad [\text{A.11}]$$

In order for the quantity  $U$  to be a minimum, the first order variation  $\delta U$  must be zero. †  $\delta U$  in [A.11] can only be zero if each term in the curly brackets is zero, so

$$\frac{\partial n}{\partial x} = \frac{d}{ds} \left[ n \frac{dx}{ds} \right] \quad [\text{A.12a}]$$

$$\frac{\partial n}{\partial y} = \frac{d}{ds} \left[ n \frac{dy}{ds} \right] \quad [\text{A.12b}]$$

$$\frac{\partial n}{\partial z} = \frac{d}{ds} \left[ n \frac{dz}{ds} \right] \quad [\text{A.12c}]$$

This set of equations [A.12a-c] is easily recognized as the scalar form of the vector *differential ray equation*

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = \nabla n \quad [\text{A.13}]$$

The differential ray equation is thus derived from the least action principle.

It is very important to recognize the conditions over which the differential ray equation holds. The derivation holds as long as the function  $n(x,y,z)$  is continuous, with continuous partial derivatives up to the second order in  $x$ ,  $y$  and  $z$ . When the function is discontinuous, the calculus of variations can also be used to derive the laws of refraction. ‡

† The condition  $\Delta U = 0$  actually forces  $U$  to be an extremum, i.e.  $U$  could be a minima, a maxima, or a saddle-point. In order for  $U$  to be a minima, the second order variation about  $U$  must be positive. Born<sup>2</sup> in Appendix I, Sections 9-11, pp.731-734, proves that this condition is met.

‡ Born,<sup>2</sup> Appendix I, Section 11, p.733.



$$n_2 \sin \theta_2 = n_1 \sin \theta_1$$

Furthermore, the condition of the minima  $\delta U = 0$  holds only as long as not more than one ray passes through any point of the neighborhood. This condition limits the rays to the same side of an envelope which can undergo multiple reflections and refractions. § In practical terms, however, this does not impose any limitation on the use of the differential ray equation; it merely recognizes that rays that are refracted or reflected will behave differently than rays that do not undergo the same refractions or reflections.

## A.2. The Differential Ray Equation and The Eikonal

The *eikonal* equation is the basic equation of geometrical optics. It is often written as

$$|\nabla\zeta|^2 = (n)^2 \quad [\text{A.14a}]$$

or explicitly as

$$\left[\frac{\partial\zeta}{\partial x}\right]^2 + \left[\frac{\partial\zeta}{\partial y}\right]^2 + \left[\frac{\partial\zeta}{\partial z}\right]^2 = (n)^2 \quad [\text{A.14b}]$$

The eikonal can also be written in terms of the unit vector  $\mathbf{s}$ , where  $\mathbf{s}$  is related to the position vector  $\mathbf{r}$  by the simple differential relationship  $\mathbf{s}ds = d\mathbf{r}$ .

$$\nabla\zeta = n\mathbf{s} = n\frac{d\mathbf{r}}{ds} \quad [\text{A.14c}]$$

In the above equations,  $n$  is the optical refractive index. But for the purposes of this discussion,  $n$  will be regarded as an arbitrary but continuous function. This allows the results of this section to be generalized, so that the references to geometrical optics are not necessary.

---

§ Born<sup>2</sup> discusses this condition in detail in Chapter 3, Section 3.3.2-3, pp.127-132. For example, in geometrical optics, rays that pass through a lens are contained within an envelope where the least action principle is satisfied; the optical length of all the rays within this envelope is minimized. Rays that do not pass through the lens might have shorter optical lengths, but these rays cannot be compared to rays that pass through the lens as they are not on the same side of the envelope.

In this section, it will be proved that the eikonal equation is a solution to the vector differential ray equation [A.13], reproduced below for convenience. ‡

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = \nabla n \quad [\text{A.15}]$$

The first step in the proof is to differentiate the vector form of the eikonal.

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = \frac{d}{ds} (\nabla \zeta) \quad [\text{A.16}]$$

Then, apply the Chain Rule to the right side of the equation.

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = d \frac{\mathbf{r}}{ds} \cdot \nabla (\nabla \zeta) \quad [\text{A.17}]$$

Substitute for  $d\mathbf{r}/ds$  using [A.14c]

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = \frac{1}{n} \nabla \zeta \cdot \nabla (\nabla \zeta) \quad [\text{A.18}]$$

Now, in vector calculus, the Chain Rule has the form below.

$$\nabla(\mathbf{A} \cdot \mathbf{A}) = 2\mathbf{A} \cdot \nabla(\mathbf{A})$$

So, Equation [A.18] can be rewritten as

$$\frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] = \frac{1}{2n} \nabla (\nabla \zeta \cdot \nabla \zeta) \quad [\text{A.19}]$$

But according to the eikonal equation,  $\nabla \zeta \cdot \nabla \zeta = n^2$ . Therefore,

$$\begin{aligned} \frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] &= \frac{1}{2n} \nabla (n^2) \\ \frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] &= \frac{2n}{2n} \nabla (n) \\ \frac{d}{ds} \left[ n \frac{d\mathbf{r}}{ds} \right] &= \nabla (n) \end{aligned} \quad [\text{A.20}]$$

This last equation, is, of course, the differential ray equation [A.15].

---

‡ This proof follows substantially that of Born,<sup>2</sup> Chapter 3.2, p.122.

All the pieces of the jigsaw puzzle may now be put together. The differential ray equation states that there is a ray with a trajectory that follows  $s(x, y, z)$ , where  $ds$  is the line element, and the unit vector  $\mathbf{s}$  is the direction of the ray at every point  $(x, y, z)$ . The eikonal equation, which has been shown to satisfy the differential ray equation, states that an arbitrary function  $\zeta(x, y, z)$  will satisfy the eikonal equation if  $|\nabla\zeta|^2 = n^2$ . Now, if  $\zeta(x, y, z) = \text{constant}$  is a 3-dimensional surface, then by definition, the gradient  $\nabla\zeta(x, y, z)$  is a vector that is normal or tangential to the surface at any point  $(x, y, z)$ . And since  $\nabla\zeta(x, y, z)$  is parallel to the unit vector  $\mathbf{s}$ , the surface  $\zeta(x, y, z)$  must be normal to the ray. Therefore, the eikonal and differential ray equations state that the surface  $\zeta(x, y, z)$  is normal to all the rays that satisfy the differential ray equation. The surfaces

$$\zeta(\mathbf{r}) = \text{constant} \quad [\text{A.21}]$$

are called the geometrical wave surfaces or the geometrical wave fronts, and the rays are the orthogonal trajectories of the wave surfaces.

It is worth mentioning at this point that the eikonal equation can also be derived from the electromagnetic wave equation. † The eikonal can then be used to derive the differential ray equation as shown in this section. This procedure is, in fact, the basis of geometrical optics. However, the purpose of the discussion in this appendix is to show that the same equations can be produced from the least action principle using variational and vector calculus. This provides for a more general approach. And as a result, the ray propagation equations can be applied to wide-ranging problems, such as geometrical optics, the mechanics of a moving particle, and even geometrical wave-front etching. This last problem, is of course, of particular interest in this thesis.

---

Born,<sup>2</sup> Chapter 3.1, pp.109-121.

### A.3. Discretizing the Scalar Form of the Differential Ray Equation

The set of differential equations [A.12a-c] may be simplified even further to yield a result that is more amenable to discretization. Writing out [A.12] in full, the following equations are obtained.

$$\frac{\partial n}{\partial x} = \frac{dn}{ds} \frac{dx}{ds} + n \frac{d^2x}{ds^2} \quad [\text{A.22a}]$$

$$\frac{\partial n}{\partial y} = \frac{dn}{ds} \frac{dy}{ds} + n \frac{d^2y}{ds^2} \quad [\text{A.22b}]$$

$$\frac{\partial n}{\partial z} = \frac{dn}{ds} \frac{dz}{ds} + n \frac{d^2z}{ds^2} \quad [\text{A.22c}]$$

The term  $dn/ds$  can be eliminated by using the Chain Rule.

$$\frac{dn}{ds} = \frac{\partial n}{\partial x} \frac{dx}{ds} + \frac{\partial n}{\partial y} \frac{dy}{ds} + \frac{\partial n}{\partial z} \frac{dz}{ds}$$

Substituting for  $dn/ds$  and dividing both sides by  $n$  yields the set of equations below.

$$\frac{d^2x}{ds^2} = \frac{1}{n} \frac{\partial n}{\partial x} - \frac{1}{n} \frac{\partial n}{\partial x} \frac{dx}{ds} \frac{dx}{ds} - \frac{1}{n} \frac{\partial n}{\partial y} \frac{dy}{ds} \frac{dx}{ds} - \frac{1}{n} \frac{\partial n}{\partial z} \frac{dz}{ds} \frac{dx}{ds} \quad [\text{A.23a}]$$

$$\frac{d^2y}{ds^2} = \frac{1}{n} \frac{\partial n}{\partial y} - \frac{1}{n} \frac{\partial n}{\partial x} \frac{dx}{ds} \frac{dy}{ds} - \frac{1}{n} \frac{\partial n}{\partial y} \frac{dy}{ds} \frac{dy}{ds} - \frac{1}{n} \frac{\partial n}{\partial z} \frac{dz}{ds} \frac{dy}{ds} \quad [\text{A.23b}]$$

$$\frac{d^2z}{ds^2} = \frac{1}{n} \frac{\partial n}{\partial z} - \frac{1}{n} \frac{\partial n}{\partial x} \frac{dx}{ds} \frac{dz}{ds} - \frac{1}{n} \frac{\partial n}{\partial y} \frac{dy}{ds} \frac{dz}{ds} - \frac{1}{n} \frac{\partial n}{\partial z} \frac{dz}{ds} \frac{dz}{ds} \quad [\text{A.23c}]$$

This set of ordinary differential equations is the differential solution to the least action problem. Now, in etching, the function  $n$  is related to the etch-rate  $R$  by

$$n(x, y, z) = \frac{1}{R(x, y, z)} \quad [\text{A.24}]$$

Therefore,

$$\frac{1}{n} \frac{\partial n}{\partial x} = -\frac{1}{R} \frac{\partial R}{\partial x} = -\frac{\partial}{\partial x} (\ln R)$$

Similar expressions apply for the differentiation over  $y$  and  $z$ . The set of differential equations [A.23a-c] may now be written in the following form : †

† This is the form used by Barouch et.al.<sup>3</sup>

$$\frac{d^2x}{ds^2} = -\frac{\partial}{\partial x}(\ln R) + \frac{\partial}{\partial x}(\ln R) \frac{dx}{ds} \frac{dx}{ds} + \frac{\partial}{\partial y}(\ln R) \frac{dy}{ds} \frac{dx}{ds} + \frac{\partial}{\partial z}(\ln R) \frac{dz}{ds} \frac{dx}{ds} \quad [\text{A.25a}]$$

$$\frac{d^2y}{ds^2} = -\frac{\partial}{\partial y}(\ln R) + \frac{\partial}{\partial x}(\ln R) \frac{dx}{ds} \frac{dy}{ds} + \frac{\partial}{\partial y}(\ln R) \frac{dy}{ds} \frac{dy}{ds} + \frac{\partial}{\partial z}(\ln R) \frac{dz}{ds} \frac{dy}{ds} \quad [\text{A.25b}]$$

$$\frac{d^2z}{ds^2} = -\frac{\partial}{\partial z}(\ln R) + \frac{\partial}{\partial x}(\ln R) \frac{dx}{ds} \frac{dz}{ds} + \frac{\partial}{\partial y}(\ln R) \frac{dy}{ds} \frac{dz}{ds} + \frac{\partial}{\partial z}(\ln R) \frac{dz}{ds} \frac{dz}{ds} \quad [\text{A.25c}]$$

The equations are now in a form more suited to discretization, since the partial terms

$$\frac{\partial}{\partial x}(\ln R) \quad \frac{\partial}{\partial y}(\ln R) \quad \frac{\partial}{\partial z}(\ln R)$$

can be evaluated independent of the increments  $dx$ ,  $dy$ ,  $dz$  and  $ds$ . The differential equation [A.25a] can be discretized as follows. The discretized forms of [A.25b] and [A.25c] are similar.

$$\begin{aligned} \frac{1}{\Delta s} \left[ \left[ \frac{(x_{i+1} - x_i)}{\Delta s} \right] - \left[ \frac{(x_i - x_{i-1})}{\Delta s} \right] \right] &= -\frac{\partial}{\partial x}(\ln R) + \\ &+ \frac{\partial}{\partial x}(\ln R) \frac{(x_{i+1} - x_i)}{\Delta s} \frac{(x_{i+1} - x_i)}{\Delta s} \\ &+ \frac{\partial}{\partial y}(\ln R) \frac{(y_{i+1} - y_i)}{\Delta s} \frac{(x_{i+1} - x_i)}{\Delta s} \\ &+ \frac{\partial}{\partial z}(\ln R) \frac{(z_{i+1} - z_i)}{\Delta s} \frac{(x_{i+1} - x_i)}{\Delta s} \quad [\text{A.26}] \end{aligned}$$

Given a step size  $\Delta s$ , initial values of  $x_i$ ,  $y_i$  and  $z_i$ , and the identity (from [A.4])

$$\Delta x^2 + \Delta y^2 + \Delta z^2 = \Delta s^2$$

it is possible to solve for the values of  $x_{i+1}$ ,  $y_{i+1}$  and  $z_{i+1}$  at the end of the time-step. Repeated iterations can, of course, be used to find the position of the curve  $s(x, y, z)$  at some final time.

#### A.4. Discretizing the Vector Form of the Differential Ray Equation

The vector form of the differential ray equation [A.13] can also be discretized for use in discrete multiple-time-step calculations. For the purposes of etching simulation, replace the function  $n(x, y, z)$  with the inverse of the etch-rate distribution  $R(x, y, z)$ , as in [A.24]. The

differential ray equation then becomes

$$\frac{d}{ds} \left[ \frac{1}{R(x,y,z)} \mathbf{s} \right] = \nabla \left[ \frac{1}{R(x,y,z)} \right] \quad [\text{A.27}]$$

The equation may be simplified by using the chain rule on the right side.

$$\frac{d}{ds} \left[ \frac{1}{R} \mathbf{s} \right] = -\frac{1}{R^2} \nabla(R) \quad [\text{A.28}]$$

Discretization comes next. If  $\Delta s$  is the distance etched in some time-step  $\Delta T$  in an average etch-rate  $R_{ave}$ , i.e.  $\Delta s = R_{ave} \Delta T$ , then

$$\frac{1}{\Delta s} \Delta \left[ \frac{1}{R} \mathbf{s} \right] = -\frac{1}{R_{ave}^2} \nabla(R) \quad [\text{A.29}]$$

Rearranging the terms,

$$\Delta \left[ \frac{1}{R} \mathbf{s} \right] = -\frac{1}{R_{ave}^2} \nabla(R) R_{ave} \Delta T = -\frac{1}{R_{ave}} \nabla(R) \Delta T \quad [\text{A.30}]$$

Now, if the equation above is made to apply between two points  $P_1$  and  $P_2$  an incremental distance apart, then [A.30] becomes

$$\begin{aligned} \frac{\mathbf{s}_2}{R_2} - \frac{\mathbf{s}_1}{R_1} &= -\frac{1}{R_{ave}} \nabla(R) \Delta T \\ \frac{\mathbf{s}_2}{R_2} &= -\frac{1}{R_{ave}} \nabla(R) \Delta T + \mathbf{s}_1 \frac{1}{R_1} \\ \mathbf{s}_2 &= -\frac{R_2}{R_{ave}} \nabla(R) \Delta T + \mathbf{s}_1 \frac{R_2}{R_1} \\ \mathbf{s}_2 - \mathbf{s}_1 &= -\frac{R_2}{R_{ave}} \nabla(R) \Delta T + \mathbf{s}_1 \left[ \frac{R_2}{R_1} - 1 \right] \end{aligned} \quad [\text{A.31}]$$

Now, the average etch-rate  $R_{ave}$  may be written as †

† Note that one could also write the average rate as

$$R_{ave} = 0.5(R_1 + R_2)$$

in which case the difference between the two forms is

$$R_{ave} - \frac{1}{(1/R)_{ave}} = \frac{(R_1 + R_2)}{2} - \frac{1}{\frac{1}{2} \left( \frac{1}{R_1} + \frac{1}{R_2} \right)} = \frac{1}{2} \frac{(R_1 - R_2)^2}{(R_1 + R_2)}$$

The difference is negligible as long as the difference in the etch-rates  $R_1$  and  $R_2$  is small.

$$\left[ \frac{1}{R} \right]_{ave} = \frac{1}{2} \left[ \frac{1}{R_1} + \frac{1}{R_2} \right] \quad [\text{A.32}]$$

The discrete etch-rate equation then becomes

$$\mathbf{s}_2 - \mathbf{s}_1 = -0.5 \frac{(R_1 + R_2)}{R_1} \nabla(R) \Delta T + \mathbf{s}_1 \left[ \frac{R_2}{R_1} - 1 \right] \quad [\text{A.33}]$$

This equation relates the difference in the unit vectors  $\mathbf{s}_1$  and  $\mathbf{s}_2$  to the gradient of the etch-rate. The differential ray equation in this form is used in the ray-based etch simulator described in Chapters 5-8.

It is worth examining Equation [A.33] for different conditions. If  $R_2 = R_1$  and  $\nabla R = 0$  then  $\mathbf{s}_1 = \mathbf{s}_2$  and there is no change in the direction vectors. This is to be expected; a ray should not be deflected as it travels through a region with uniform etch-rate. Now, if  $\nabla R$  is a vector parallel to  $\mathbf{s}_1$ , e.g. if  $\mathbf{s}_1$  and  $\nabla R$  have only  $z$ -components, then clearly  $\mathbf{s}_2$  will remain parallel to  $\mathbf{s}_1$ . Physically, this means that a ray traveling normal to a velocity field will not be refracted. In optics, a light ray perpendicular to a glass plate will not be deflected; the only effect of the glass plate is to slow down the speed of the ray.

What happens if  $|\nabla R| \gg 1$ ? This could occur if the etch-rate is discontinuous, such as at an abrupt interface. From Equation [A.33], there is clearly a limit on the magnitude of the vectors. Taking the magnitude of the vectors in [A.33] yields

$$\begin{aligned} \mathbf{s}_2 - \mathbf{s}_1 &= \Delta \mathbf{s} \\ |\mathbf{s}_2 - \mathbf{s}_1|^2 &= |\Delta \mathbf{s}|^2 \end{aligned} \quad [\text{A.34}]$$

Since  $\mathbf{s}_1$  and  $\mathbf{s}_2$  are unit vectors, there is a size-limitation on  $\Delta \mathbf{s}$ , i.e.

$$|\Delta \mathbf{s}| < 2 \quad [\text{A.35}]$$

since the maximum length of two unit vectors is 2. Therefore, if  $|\nabla R| \gg 1$  or if  $\Delta T$  is very large, then the term on the left of Equation [A.33] could have a magnitude larger than 2, which violates [A.35]. This means that the differential ray equation will not work at abrupt

boundaries or when the time-step chosen is too large. At abrupt boundaries, it is necessary to use Snell's law of refraction. For large time-steps, if  $|\Delta s| > 2$ , the time-step has to be broken up into smaller time-steps. This leads to the notion of recursive length checking, discussed in Chapter 7.



## REFERENCES

1. L.B. Carll, in *A Treatise on the Calculus of Variations*, pp. 335-344, John Wiley & Sons, New York, 1881. Chapter 1, Section X.
2. M. Born, E. Wolf, in *Principles of Optics, Sixth Edition*, Pergamon Press, London 1980.
3. E. Barouch, B. Bradie, S.V. Babu, "Calculation of Developed Resist Profiles by Least Action Principle," *Interface'88 : Proceedings of KTI Microelectronics Seminar*, pp. 187-196, November 1988. 2D Ray method.

## APPENDIX B

### APPENDIX B USER GUIDES

## NAME

sample3D - 3D Simulation of Photoresist Development

## SYNOPSIS

```
sample3D [-if splat-inputfile] [-cf contourfile] [-bf bleach-inputfile] [-solid] [-ascii] [-rb
binary-etchrate-file] [-ra ascii-etchrate-file] [-et etchtime] [-pt printtime] [-plot] [-h]
```

## DESCRIPTION

**SAMPLE3D** is a user-oriented shell script for the Simulation and Modeling of Profiles in Lithography and Etching. The program integrates a number of process simulators for three-dimensional simulation of optical lithography, while also providing display and print capabilities.

The simulation of optical lithography involves modeling the process by which patterns on a mask are transferred onto a photoresist-coated wafer via exposure to optical radiation. Essentially, this process involves three major components : imaging, exposure-bleaching and etching. **SAMPLE3D** simulates the three-dimensional (3D) profile of the developed photoresist as a function of time by first using **SPLAT** to calculate the aerial image intensity incident upon the photoresist. The exposure of the photoresist to light triggers chemical changes in the photoresist; the chemical changes or exposure-bleaching is modeled using Dill's algorithm in **BLEACH**. An exposure model then generates a three-dimensional etch-rate distribution throughout the volume of the photoresist. This distribution is then used in **ETCH**, a three-dimensional etch simulator based on the ray-string algorithm, to generate a three-dimensional profile of the photoresist.

The options for each of the process modules **SPLAT**, **BLEACH** and **ETCH**, are specified through the command line. In order to run **SPLAT**, the *splat-inputfile* must be specified. (See the *SPLAT User Manual* for details on **SPLAT** input formats.) Similarly, to run **BLEACH** for exposure-bleaching simulation, the *bleach-inputfile* must be specified. This inputfile is similar in format to that of **SAMPLE**. (See the *SAMPLE User Guide* for details.) **BLEACH** will produce a three-dimensional etch-rate array for use in **ETCH**. However, **ETCH** will only run if the *etch-time* has been specified in the command line.

A full 3D simulation involving all three modules can be run using the command :

```
% sample3D -if splat-inputfile -bf bleach-inputfile -et 30
```

where *splat-inputfile* and *bleach-inputfile* are the names of the input files. The command above will run the etch simulator for 30 seconds of development time.

Alternately, **SAMPLE3D** can be run without using all the simulation modules mentioned previously. For example

```
% sample3D -cf contour-file -bf bleach-inputfile
```

will only run **BLEACH**. **BLEACH** will in turn produce a binary etch-rate file, "rval.3D.binary". **ETCH** can then be run using the following command.

```
% sample3D -rb rval.3D.binary -et 30 -pt 10
```

Using commands such as the above, it is no longer necessary to run full 3D simulations. This saves time, since the exposure-bleaching simulation can be quite time-consuming if the process involves post-exposure bake diffusion.

## OPTIONS

**-if *splat-inputfile***

reads the *SPLAT* input data from the file *splat-inputfile*. (See *SPLAT User Manual* for detailed instructions.) *SPLAT* will only be run if the *splat-inputfile* is specified. If a *contour-file* is defined using *SPLAT*'s Trial 11, then the script will pass on the *contour-file* to *BLEACH*.

**-cf *contour-input-file***

specifies the file in which the intensity contour data is stored. This command is used to bypass *SPLAT*. If both the *contour-file* and the *bleach-inputfile* are specified, the script will branch to the *BLEACH* simulator. The contour data must be arranged as specified in the *contour* manual.

**-bf *bleach-inputfile***

reads the exposure-bleach instructions from the file *bleach-inputfile*. This file has the exact same format as that of the (2d) *SAMPLE* program. (See *SAMPLE 1.7a User Guide* for detailed instructions.) *BLEACH* will only run if both the *contour-file* and the *bleach-inputfile* are specified. If the *etch-time* is specified as well, the script will branch to the *ETCH* simulator for the etching simulations.

**-solid**

will cause the *BLEACH* program to print out 3D contours of constant M-values to the file "mxyz.solid.3D". The data is printed out in *pdraw* format, and can be plotted out using the command

```
% pdraw -h -nosort mxyz.solid.3D
```

**-ascii**

forces the *BLEACH* simulator to print out the etch-rate data in ascii format. The ascii file is "rval.3D". This is useful if the data-file is to be ported to a different machine. The default, however, (if *-ascii* is not specified) is to print the data in a binary file, "rval.3D.binary". Reading and writing in binary significantly reduces the I/O time.

**-rb *binary-etchrate-file***

will force the program to read the etch-rate data from the binary file *binary-etchrate-file*. This option will bypass both *SPLAT* and *BLEACH*. However, *ETCH* will only run if the *etch-time* has been specified.

**-ra *ascii-etchrate-file***

will force the program to read the etch-rate data from the ascii file *ascii-etchrate-file*. This option will bypass both *SPLAT* and *BLEACH*. However, *ETCH* will only run if the *etch-time* has been specified.

**-et etchtime****-pt printtime**

specifies the total etch and print times for the *ETCH* simulation program. The *etch-time* specifies the total development time, and the profile will be plotted every *printtime* seconds. Note : *ETCH* will not run unless the *etch-time* has been specified.

**-plot**

causes interactive plotting.

**-h**

forces the *ETCH*-simulated plots to be drawn with hiddenlines.

## AUTHOR

Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

## FILES

<i>mxyz.solid.3D</i>	contours of constant M
<i>rval.3D</i>	ascii etch-rate data (3D array)
<i>rval.3D.binary</i>	binary etch-rate data (3D array)
<i>curves.plot</i>	3D photoresist profile
<i>curves.plot.2D</i>	3D photoresist profile projected onto the x-y plane
<i>dataplot.ps</i>	temporary POSTSCRIPT file

## SEE ALSO

*drawplot(L)*, *contour(L)*, *pdraw(L)*  
*SPLAT* User Manual, *BLEACH(L)*, *ETCH(L)*  
*SAMPLE* User Guide, *parse\_splat(L)*

**NAME**

**pdraw** - 3D plot program for X-windows and Postscript

**SYNOPSIS**

**pdraw** [-v vx vy vz] [-o *options-file*] [-P*printer*] [-s *scale*] [-e] [-h] [-nosort] [-noplots] [-print] [-ps] [-tl "*toplabel*"] [-xl "*xlabel*"] [-yl "*ylabel*"] [-zl "*zlabel*"] *plotfile1 plotfile2...*

**DESCRIPTIONS**

**Pdraw** is a program for drawing 3D plots on X10 or X11 windows. The program will also produce a POSTSCRIPT plot which can be dumped out to an APPLE Laserwriter. **Pdraw** reads in x-y-z data from a *plot-file* and manipulates that data according to options specified either in the command-line or in a *options-file*. The *plot-file* can be compressed (see `compress(1)`); compressed files will be uncompressed automatically. The program then plots lines on a screen or dumps the plots to a POSTSCRIPT file.

The *plot-file* input data consists of alternating x, y and z values, in the format shown below.

**Data File Format (plot-file)**

```
xmin xmax ymin ymax zmin zmax
ncurves
npts
x1 y1 z1
x2 y2 z2
..
..
..
npts
x1 y1 z1
x2 y2 z2
..
..
..
```

In the above, *xmin*, *xmax*, *ymin*, *ymax*, *zmin* and *zmax* are lower and upper bounds of the desired plot, *ncurves* are the number of curves to be plotted, and *npts* are the number of points in each curve. The data file can consist of more than one set of curves to be plotted; each set (i.e. one set for each separate graph) is separated from the next by a blank line.

Upon starting up the program, **pdraw** will read in the data stored in the *plot-file*, as well as any plotting options specified either in the command line or in the *options-file*. **Pdraw** then uses the given view direction to rotate and transform the plot onto a plane perpendicular to the viewing vector. Currently, only parallel projection is supported. If the program is being run under X-windows, the plot will then be drawn on the screen. The viewing vector can be changed using the "H", "J", "K", "L" and "O" keys on the the keyboards; the plot on the screen can be rotated sideways using the "H" or "L" keys, rotated up or down using the "J" and "K" keys, and drawn with the original viewing vector using "O". The "A", "S", "D" and "F" keys will produce 90° rotations. "Z" will plot the image projected on the x-y plane ( $z=\text{constant}$ ),

"Y" will plot the image projected on the x-z plane ( $y=\text{constant}$ ), and "X" will plot the image projected on the y-z plane ( $x=\text{constant}$ ). The final view angle will be saved and used for the POSTSCRIPT plot. Finally, the user will be prompted as to whether or not the POSTSCRIPT plot is to be sent to a printer.

## OPTIONS

### -v vx vy vz

reads in the viewing eye position, relative to (0,0,0). The plot will be rotated and transformed so that the z-axis is parallel to this position. For example, a view position of (1,0,0) means that the 3D structure is being viewed with parallel projection from the x-axis.

### -o options-file

reads plotting options from the file *options-file*. Each option specification consists of a keyword and its corresponding value. The parser recognizes only a limited set of keywords; their values are either numbers, quoted strings, or the words "on" and "off". All the words in the option specification must be on the same line. The pound sign (#) indicates that the remainder of the line is a comment to be ignored by the parser.

#### List of Options (options-file)

xlabel "LABEL"	#[default = "X-Axis"]	- for the x-label
ylabel "LABEL"	#[default = "Y-Axis"]	- for the y-label
zlabel "LABEL"	#[default = "Z-Axis"]	- for the z-label
toplabel "LABEL"	#[default = "3D Line Plot"]	- for the top-label
equalscale on/off	#[default = on]	- for equal x-y scaling
postscript on/off	#[default = on]	- for postscript (PS) plot
printplot on/off	#[default = off]	- send PS file to printer
noplot on/off	#[default = off]	- no graphics plot
printer "PRINTER"	#[default = \$PRINTER]	- define the printer
line on/off	#[default = on]	- draw the line
linechange on/off	#[default = off]	- change the linetypes
marker on/off	#[default = off]	- draw the marker
markerchange on/off	#[default = off]	- change the markertypes
hiddenline on/off	#[default = off]	- for hidden-line drawings
nosort on/off	#[default = off]	- for hidden-line drawings
scale [0.1 - 1.0]	#[default = 1.00]	- scales the PS plot
xticks [1 - 20]	#[default = 2]	- no. of x-divisions
yticks [1 - 20]	#[default = 2]	- no. of y-divisions
zticks [1 - 20]	#[default = 2]	- no. of z-divisions

### -Pprinter

specifies which printer to which to send the postscript plot. The current default sets the printer name to the environment variable \$PRINTER. If this variable is not set, then the printer used is the lp550M printer in 550M Cory.

### -s scale

sets a scale factor. This is used only for POSTSCRIPT plotting.

-e forces *unequal* scales to be applied to the x, y and z axes. The boundary of the 3D object will then resemble a cube.

- h** draws polygons with hidden-lines.
- nosort**  
prevents sorting for the hidden-line option.
- noplots**  
prevents plots on the graphics display.
- print**  
sends plots to the printer automatically.
- ps** turns the postscript plotting option off. This can also be done by setting the POSTSCRIPT environment variable to OFF. e.g. % setenv POSTSCRIPT off
- tl toplevel**
- xl xlabel**
- yl ylabel**
- zl zlabel**  
sets label options.
- host:display**  
opens a window on the given host and display
- =geom**
- rv**
- bw border-width**
- bd color**
- fg color**
- bg color**  
sets input options for the X-window system.

## BUGS

Not really that many. The POSTSCRIPT labels need to be adjusted. The labels don't come out well when the picture is rotated beyond the default view. The parser needs to be improved. There should be a better way to put change linetypes and markertypes. Log axes might be nice. Also should incorporate drawplot modifications here.

## AUTHOR

Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

## FILES

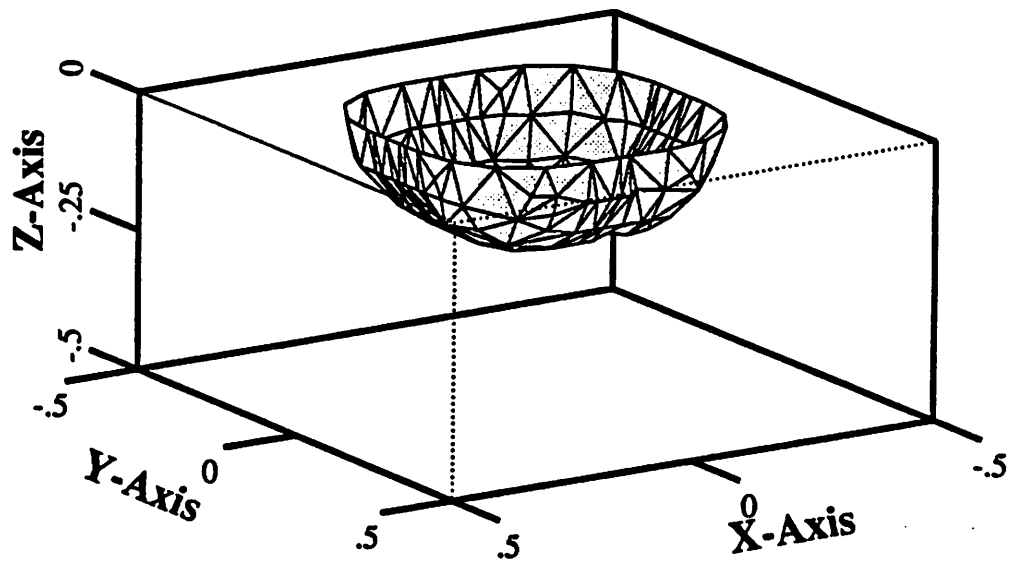
*dataplot.ps* temporary POSTSCRIPT file

## SEE ALSO

contour(L), drawplot(L)



### 3D Line Plot



## NAME

contour - contour plot program for X-windows, HP2648s and Postscript

## SYNOPSIS

contour [-o *options-file*] [-P*printer*] [-s *scale*] [-c *level*] [-cstep *step-size*] [-ex] [-g] [-j *join-level*] [-l] [-noplot] [-print] [-ps] [-old] [-tl "*toplabel*"] [-xl "*xlabel*"] [-yl "*ylabel*"] [-3D] *contour-file*

## DESCRIPTIONS

Contour is a program for drawing contour plots on X10/X11 windows or HP2648 terminals. The program will also produce a POSTSCRIPT plot which can be dumped out to an APPLE Laserwriter. Contour reads in data on a 3D surface from a *contour-file* and manipulates that data according to options specified either in the command-line or in a *options-file*. The *plot-file* can be compressed (see `compress(1)`); compressed files will be uncompressed automatically. The program then draws contours on a screen or dumps the contours to a POSTSCRIPT file.

The 3D surface input data consists of z-values of the 3D surface, arranged on a rectangular grid of size  $(x_{max} - x_{min}) \times (y_{max} - y_{min})$ . The data file format is shown below.

Data File Format (*contour-file*)

```
xmin xmax ymin ymax
nxpts nypts
z1
z2
..
..
..
```

In the above, *xmin*, *xmax*, *ymin* and *ymax* are lower and upper bounds of the grid, and *nxpts* and *nypts* are the number of grid divisions in x and y. Alternately, if the -3D flag is specified, the data-file could consist of triangles, specified in `pdraw(L)` format, i.e.,

Data File Format (*plot-file*)

```
xmin xmax ymin ymax zmin zmax
ncurves
4.0
x1 y1 z1
x2 y2 z2
x3 y3 z3
x1 y1 z1
4.0
x1 y1 z1
x2 y2 z2
x3 y3 z3
x1 y1 z1
```

..  
..

Upon starting up the program, *contour* will read in the data stored in the *contour-file* and will then find the maximum and minimum z-values of the surface. It will then prompt for a contour step-size (i.e. the contour increments), and read in any plotting options specified either in the command line or in the *options-file*. The plot will then be drawn on the screen if possible. Finally, the user will be prompted as to whether or not the POSTSCRIPT plot is to be sent to a printer.

## OPTIONS

### -o *options-file*

reads plotting options from the file *options-file*. Each option specification consists of a keyword and its corresponding value. The parser recognizes only a limited set of keywords; their values are either numbers, quoted strings, or the words "on" and "off". All the words in the option specification must be on the same line. The pound sign (#) indicates that the remainder of the line is a comment to be ignored by the parser.

#### List of Options (*options-file*)

xlabel "LABEL"	#[default = "X-AXIS"]	- for the x-label
ylabel "LABEL"	#[default = "Y-AXIS"]	- for the y-label
toplabel "LABEL"	#[default = "CONTOUR PLOT"]	- for the top-label
grid on/off	#[default = off]	- draws a grid
equalscale on/off	#[default = on]	- for equal x-y scaling
noplot on/off	#[default = off]	- don't draw graphics plot
postscript on/off	#[default = on]	- for postscript (PS) plot
printplot on/off	#[default = off]	- print PS file automatically
printer "PRINTER"	#[default = \$PRINTER]	- define the printer
contlabel on/off	#[default = on]	- for contour labels
joinlevel high/low	#[default = --]	- for joining curves
scale [0.1 - 1.0]	#[default = 1.00]	- scales the PS plot
linetypes [1 - 3]	#[default = 2]	- no. of contour linetypes
xticks [1 - 20]	#[default = 4]	- no. of x-divisions
yticks [1 - 20]	#[default = 4]	- no. of y-divisions

### -P*printer*

specifies which printer to which to send the postscript plot. The current default sets the printer name to the environment variable \$PRINTER. If this variable is not set, then the printer used is the lp550M printer in 550M Cory.

### -s *scale*

sets a scale factor. This is used only for POSTSCRIPT plotting.

### -c *level*

forces the program to compute the contours at a single value of z, specified by *level*. The contours will be written to the file *image.cont*. The output data is organized in SAMPLE plot format, i.e.,

**Single contour (image.cont)**

```

xmin, xmax, ymin, ymax
ncurves
npts
pt1.x pt1.y
pt2.x pt2.y
..
..
..
npts
pt1.x pt1.y
..
..

```

- cstep *step-size***  
defines the contour step-size. If this is not defined, the program prompts for the step-size.
- ex** expands the data into a triangular mesh. The mesh is stored in *contour-file.3D*. Thus, if the initial contour file is named *im.cont*, then the surface mesh will be stored in *im.cont.3D*.
- g** forces a grid to be drawn.
- j *joinlevel***  
causes contour curves to be joined where possible. This is done by defining a boundary layer around the rectangular border, and setting the z-value of that boundary layer at either the maximum or minimum z-value. *joinlevel = HIGH* or *high* sets the border z-value to its maximum value: this is useful for plots which have high average z-values. *joinlevel = LOW* or *low* sets the border z-value to its minimum value: this is useful for plots which have low average z-values.
- l** suppresses the contour labels.
- noplot**  
prevents plots on the graphics display.
- print**  
sends the POSTSCRIPT lot to the printer automatically.
- ps** turns off the postscript plotting mode. This can also be done by setting the POSTSCRIPT environment variable to OFF. e.g. % setenv POSTSCRIPT off
- old** accepts an older contour format for the *contour-file*, based on a rectangular  $50 \times 50$  array. The data file format is similar to that described earlier except for the first 3 lines. *nxpts* and *nypts* (both equal to 50) are omitted. Also, *xmin* and *ymin* are the coordinates of the lower left corner of the grid while *xlength* and *ylength* define the area being examined.

**Old Data File Format (contour-file)**

```

xmin ymin xlength ylength
z1
z2
..

```

..  
..

**-tl toplabel**  
**-xl xlabel**  
**-yl ylabel**  
    sets label options.

**-3D** specifies the plotfile to be of **pdraw(L)** format. As such, the plotfile should consist of triangles only. For example, "contour -3D curves.plot.3D" will produce contour plots of the 3D plotfile *curves.plot.3D*.

**host:display**  
    opens a window on the given host and display

**-d host:display**  
**=geom**  
**-rv**  
**-bw border-width**  
**-bd color**  
**-fg color**  
**-bg color**  
**-fn font-name**  
    sets input options for the X-window system.

#### AUTHOR

Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

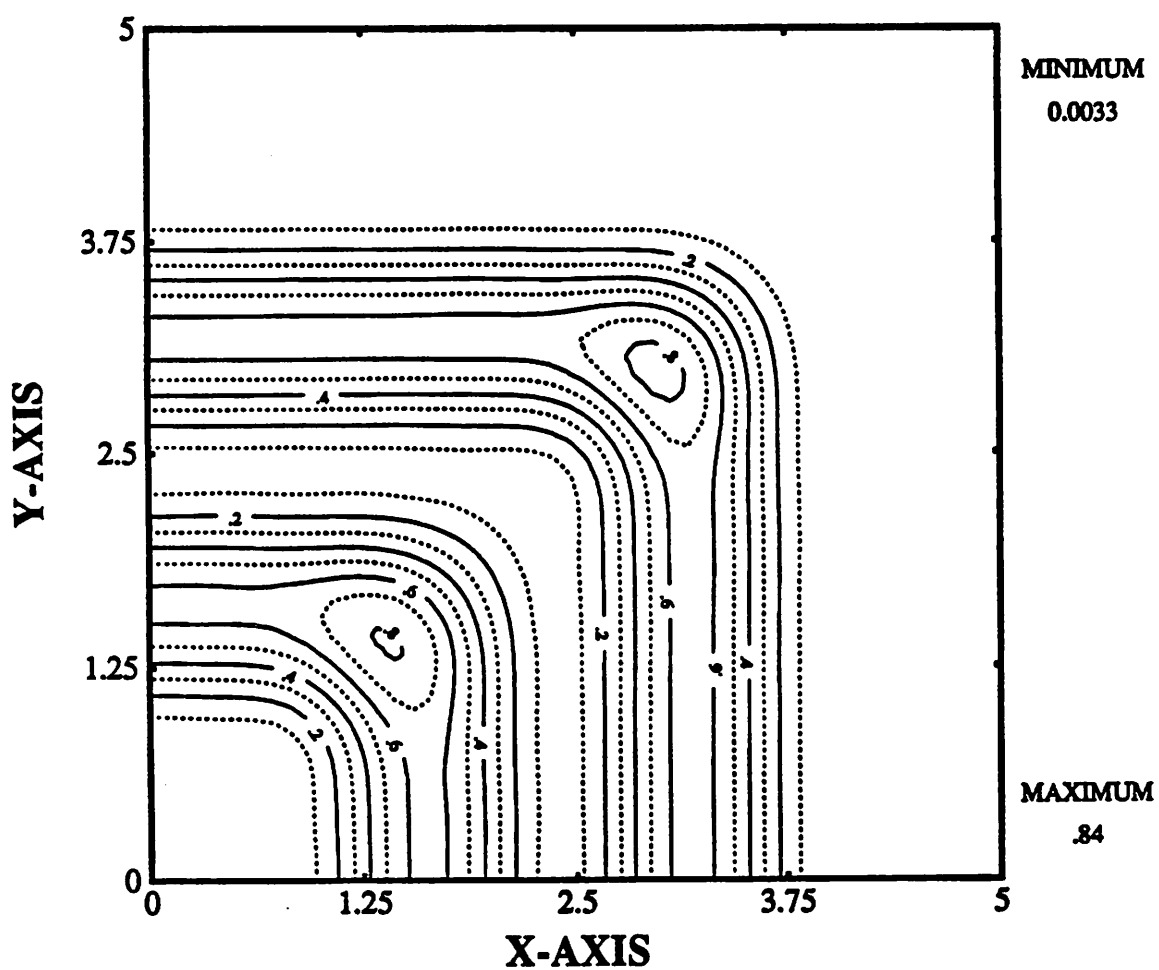
#### FILES

*dataplot.ps* temporary POSTSCRIPT file  
*image.cont* file produced by **-c level** option

#### SEE ALSO

SPLAT, drawplot(L), pdraw(L)

# CONTOUR PLOT



## NAME

drawplot - plot program for Sunview, X-windows, HP2648s and Postscript

## SYNOPSIS

```
drawplot [-o options-file] [-Pprinter] [-s scale] [-ar] [-clip] [-e] [-g] [-l] [-noplot] [-print] [-ps] [-xlog] [-ylog] [-t template-file] [-tl "toplabel"] [-xl "xlabel"] [-yl "ylabel"] plot-file1 plot-file2...
```

## DESCRIPTIONS

Drawplot is a program for drawing 2D plots on X10/X11 windows, SUNVIEW displays, or HP2648 terminals. The program will also produce a POSTSCRIPT plot which can be dumped out to an APPLE Laserwriter. Drawplot reads in x-y data from a *plot-file* and manipulates that data according to options specified either in the command-line or in a *options-file*. The *plot-file* can be compressed (see `compress(1)`); compressed files will be uncompressed automatically. The program then plots lines on a screen or dumps the plots to a POSTSCRIPT file.

The *plot-file* input data consists of alternating x and y values, in the format shown below.

## Data File Format (plot-file)

```
xmin xmax ymin ymax
ncurves
npts
x1 y1
x2 y2
..
..
..
npts
x1 y1
x2 y2
..
..
..
```

In the above, *xmin*, *xmax*, *ymin* and *ymax* are lower and upper bounds of the desired plot, *ncurves* are the number of curves to be plotted, and *npts* are the number of points in each curve. The data file can consist of more than one set of curves to be plotted; each set (i.e. one set for each separate graph) is separated from the next by a blank line.

Upon starting up the program, drawplot will read in the data stored in the *plot-file*, as well as any plotting options specified either in the command line or in the *options-file*. The plot will then be drawn on the screen if possible. Finally, the user will be prompted as to whether or not the POSTSCRIPT plot is to be sent to a printer.

## OPTIONS

**-o options-file**

reads plotting options from the file *options-file*. Each option specification consists of a keyword and its corresponding value. The parser recognizes only a limited set of keywords; their values are either numbers, quoted strings, or the words "on" and "off". All the words in the option specification must be on the same line. The pound sign (#) indicates that the remainder of the line is a comment to be ignored by the parser.

**List of Options (options-file)**

xlabel "LABEL"	#[default = "X-AXIS"]	- for the x-label
ylabel "LABEL"	#[default = "Y-AXIS"]	- for the y-label
toplabel "LABEL"	#[default = "X-Y LINE PLOT"]	- for the top-label
autorange on/off	#[default = on]	- automatic axis ranging
clip on/off	#[default = off]	- clips the picture
equalscale on/off	#[default = off]	- for equal x-y scaling
grid on/off	#[default = off]	- draws a grid
noplot on/off	#[default = off]	- don't draw graphics plot
postscript on/off	#[default = on]	- for postscript (PS) plot
printplot on/off	#[default = off]	- print PS file automatically
printer "PRINTER"	#[default = \$PRINTER]	- define the printer
line on/off	#[default = on]	- draw the line
linechange on/off	#[default = off]	- change the linetypes
marker on/off	#[default = off]	- draw the marker
markerchange on/off	#[default = off]	- change the markertypes
landscape on/off	#[default = off]	- print in landscape mode
xlog on/off	#[default = off]	- x-axis in log-scale
ylog on/off	#[default = off]	- y-axis in log-scale
scale [0.1 - 1.0]	#[default = 1.00]	- scales the PS plot
xticks [1 - 20]	#[default = 4]	- no. of x-divisions
yticks [1 - 20]	#[default = 4]	- no. of y-divisions
sleeptime [1 - 20]	#[default = 5]	- SUNVIEW plot time
templatefile "template"	#[default = ""]	- plot/marker templates

**-Pprinter**

specifies which printer to which to send the postscript plot. The current default sets the printer name to the environment variable \$PRINTER. If this variable is not set, then the printer used is the lp550M printer in 550M Cory.

**-s scale**

sets a scale factor. This is used only for POSTSCRIPT plotting.

**-ar** forces no automatic ranging of the x and y axes. Without this option, automatic ranging is set ON, and the program tries to select the best axis-scales possible. The autorange option is also turned off when either *xticks* or *yticks* is defined in the *option-file*.

**-clip** forces clipping within the plot boundary.

**-e** causes the x and y grids to be drawn with equal scales.

**-g** forces the grid to be drawn.



- l prints the postscript plot in landscape mode (rotated).
- noplots stops the program from drawing on a SUN, X or HP graphics window. Only the postscript plot will be made.
- print sends the postscript plot to the printer automatically.
- ps turns off the postscript plotting mode. This can also be done by setting the POSTSCRIPT environment variable to OFF. e.g. % setenv POSTSCRIPT off
- xlog draws the x-axis on a log scale.
- ylog draws the y-axis on a log scale.
- t *template-file* reads the plot and marker template from the file *template-file*. This is used to set the line type, marker type, and line-label of each line. The file should have 2 numbers per line; the first number corresponds to the line type [0-10], while the second corresponds to the marker type [0-16]. The line-label is the first word that follows the keywords "label", "linelabel" or "line\_label". The line-label should be placed inside quotes. The number of lines in the *template-file* corresponds to the number of different line or marker types that will appear in the plot - the curves will cycle through these plot/marker types. A sample *template-file* is shown below.

#### Example Plot/Marker Template File (template-file)

ln_type = 1	mkr_type = 0	label = "Line 1"
ln_type = 2	mkr_type = 0	label = "Line 2"
ln_type = 3	mkr_type = 1	label = "Line 3"
ln_type = 4	mkr_type = 13	label = "Line 4"
ln_type = 5	mkr_type = 2	label = "Line 5"
ln_type = 6	mkr_type = 4	label = "Line 6"

#### POSTSCRIPT Line Types (example : % drawplot -t template.ln f77lines)

0	No Line
1	Solid Line
2	Dashed Line
3	Dotted Line
4	Dot Dash
5	Double Dot
6	Long Dash
7	Dot Dash
8	Long Dots
9	Short Dash
10	Dot-Dot Dash

#### POSTSCRIPT Marker Types (example : % drawplot -t template.mkr f77lines)

0	No Marker
1	Point
2	Square (White)
3	Square (Gray)
4	Square (Black)
5	Diamond (White)
6	Diamond (Gray)
7	Diamond (Black)
8	Upright Triangle (White)
9	Upright Triangle (Gray)
10	Upright Triangle (Black)
11	Upsidedown Triangle (White)
12	Upsidedown Triangle (Gray)
13	Upsidedown Triangle (Black)
14	Circle (White)
15	Circle (Gray)
16	Circle (Black)
17	X marks the spot.

**-tl toplabel**

**-xl xlabel**

**-yl ylabel**

sets label options.

**host:display**

opens a window on the given host and display

**-d host:display**

**=geom**

**-rv**

**-bw border-width**

**-bd color**

**-fg color**

**-bg color**

**-fn font-name**

sets input options for the X-window system.

## AUTHOR

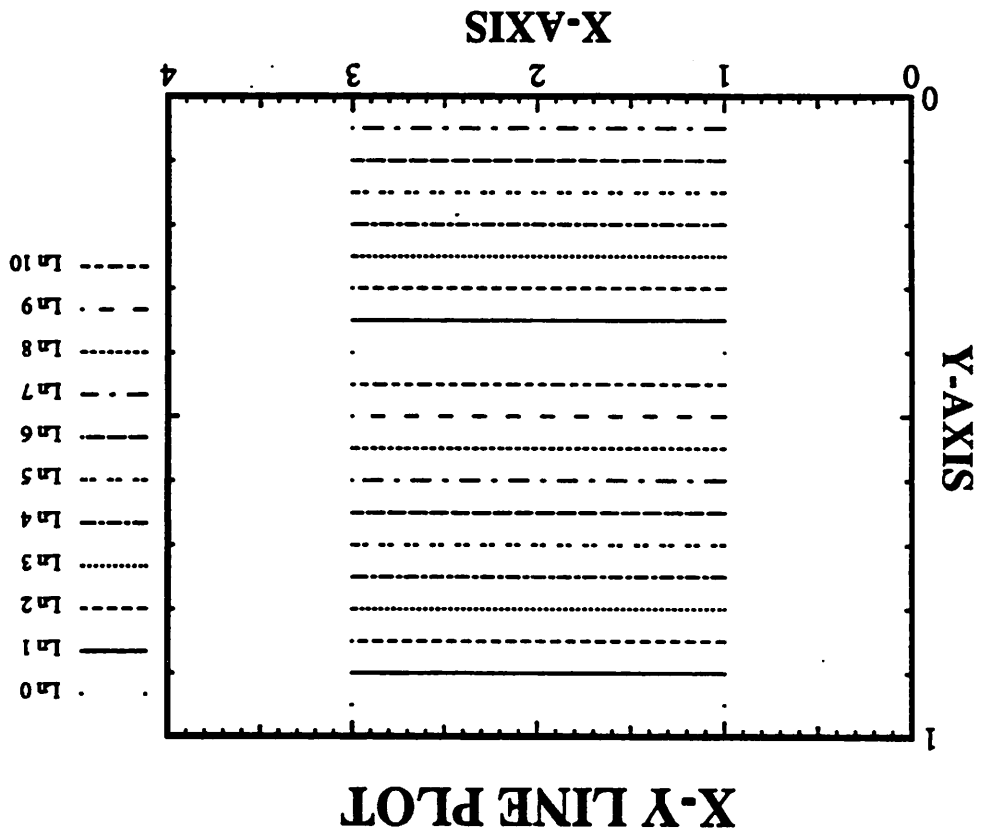
Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

## FILES

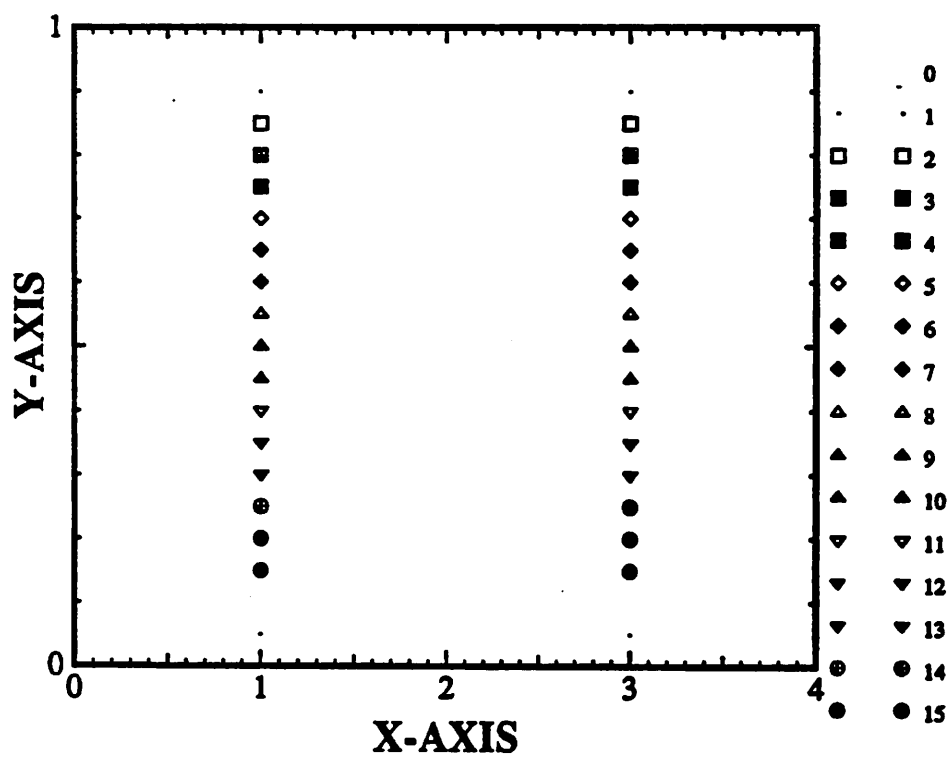
*dataplot.ps* temporary POSTSCRIPT file

**SEE ALSO**

**contour(L), pdraw(L)**



## X-Y LINE PLOT



## NAME

**drawmask** - draw an X11 and POSTSCRIPT plot of a SPLAT mask

## SYNOPSIS

**drawmask** [-o *options-file*] [-P*printer*] [-l] [-lbl] [-s *scale*] [-noplot] [-ps] [-print] [-tl "toplabel"] [-xl "xlabel"] [-yl "ylabel"] *mask-file*

## DESCRIPTIONS

**Drawmask** is a program that draws an X11 and POSTSCRIPT plot of a SPLAT mask. **Drawmask** reads in mask data from a *mask-file* and manipulates that data according to options specified either in the command-line or in a *options-file*. The program then dumps the plot to a POSTSCRIPT file.

## OPTIONS

**-o *options-file***

reads plotting options from the file *options-file*. Each option specification consists of a keyword and its corresponding value. The parser recognizes only a limited set of keywords; their values are either numbers, quoted strings, or the words "on" and "off". All the words in the option specification must be on the same line. The pound sign (#) indicates that the remainder of the line is a comment to be ignored by the parser.

**List of Options (options-file)**

xlabel "LABEL"	#[default = "X-AXIS"]	- for the x-label
ylabel "LABEL"	#[default = "Y-AXIS"]	- for the y-label
toplabel "LABEL"	#[default = "2D Mask"]	- for the top-label
equalscale on/off	#[default = on]	- for equal x-y scaling
noplot on/off	#[default = off]	- don't draw graphics plot
postscript on/off	#[default = on]	- print PS file automatically
printplot on/off	#[default = off]	- send PS file to printer
printer "PRINTER"	#[default = \$PRINTER]	- define the printer
landscape on/off	#[default = off]	- print in landscape mode
scale [0.1 - 1.0]	#[default = 1.00]	- scales the PS plot
xticks [1 - 20]	#[default = 4]	- no. of x-divisions
yticks [1 - 20]	#[default = 4]	- no. of y-divisions
nolabel on/off	#[default = on]	- don't print labels

**-P*printer***

specifies which printer to which to send the postscript plot. The current default sets the printer name to the environment variable \$PRINTER. If this variable is not set, then the printer used is the lp550M printer in 550M Cory.

**-l** prints the postscript plot in landscape mode (rotated).

**-lbl** prints the mask with labels.

**-s *scale***

sets a scale factor. This is used only for POSTSCRIPT plotting.

**-noplots**

prevents plots on the graphics display.

**-print**

sends the POSTSCRIPT lot to the printer automatically.

**-ps** turns off the postscript plotting mode. This can also be done by setting the POSTSCRIPT environment variable to OFF. e.g. % setenv POSTSCRIPT off

## BUGS

Currently, the program is only able to plot opaque masks (transmission = 0) with rectangular openings (no triangles accepted). There is no rectangle-intersection checking.

## AUTHOR

Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

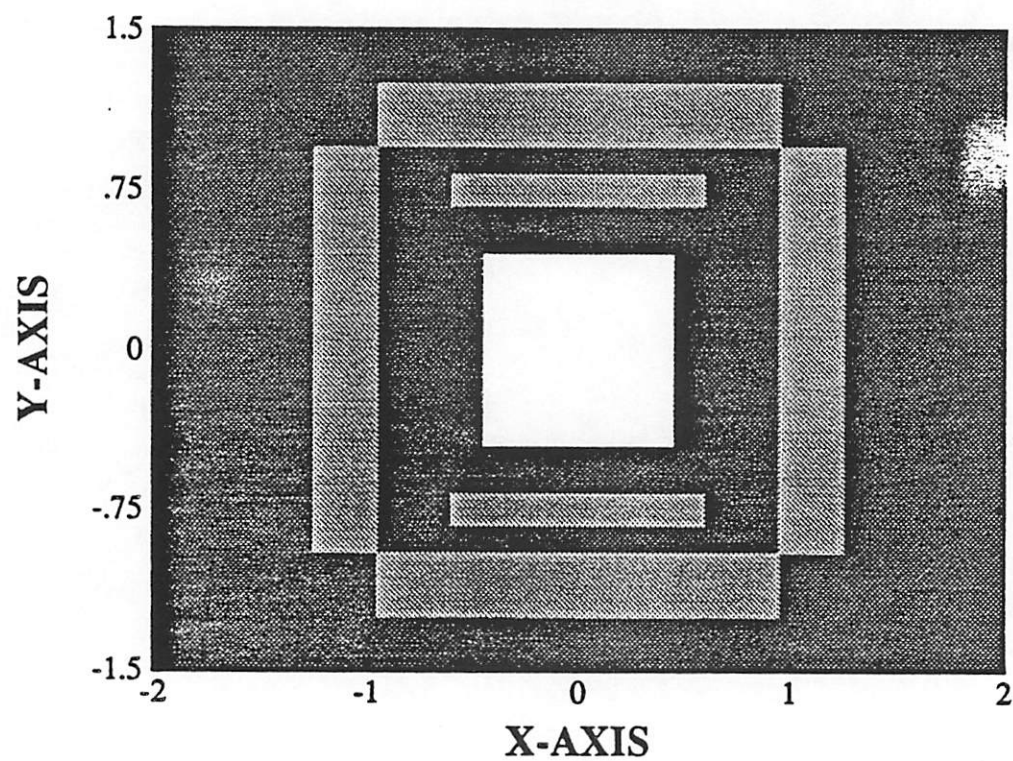
## FILES

*dataplot.ps* temporary POSTSCRIPT file

## SEE ALSO

SPLAT, drawplot(L), contour(L)

## 2D Mask





**NAME**

conv - convert a 3D contour format data-file to pdraw format

**SYNOPSIS**

conv [-t] [-old] [-o *output-file*] *input-file*

**DESCRIPTION**

Conv is a program for converting a contour data-file to a pdraw data-file. The pdraw output-file will be either a rectangular wire-mesh or a triangular solid-mesh (-t flag). If the name of the *output-file* is not specified in the command line, the *output-file* will be named *input-file.3D* (i.e. the ".3D" will be appended to the *input-file*). The *contour* data-file can be compressed (see `compress(1)`); compressed files will be uncompressed automatically.

See `contour (L)` and `pdraw (L)` for details on the input/output data formats.

**OPTIONS**

- t creates a triangular mesh. This is better for hidden-line plots.
- old accepts an older contour format for the *contour-file*, based on a rectangular 50 × 50 array. See `contour (L)` for more details.
- o *output-file*  
specifies the name of the output-file.

**AUTHOR**

Kenny K.H. Toh (ktoh@mascot.berkeley.edu)

**SEE ALSO**

drawplot(L), contour (L), pdraw(L)