CENTRALIZING GEOMETRY SERVICES FOR THREE-DIMENSIONAL INTEGRATED CIRCUITS TOPOGRAPHY SIMULATION

by

Robert Hsiung-Fu Wang

Memorandum No. UCB/ERL M95/102

15 December 1995

COVER PAGE

CENTRALIZING GEOMETRY SERVICES FOR THREE-DIMENSIONAL INTEGRATED CIRCUITS TOPOGRAPHY SIMULATION

by

Robert Hsiung-Fu Wang

Memorandum No. UCB/ERL M95/102

15 December 1995

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Centralizing Geometry Services for Three-Dimensional Integrated Circuits Topography Simulation

by

Robert Hsiung-Fu Wang

Doctor of Philosophy in

Engineering - Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor Andrew R. Neureuther, Chair

This thesis initiates research in a new field of centralizing geometry services for 3D IC topography simulation, and contributes organizational approaches, performance testing methodology, and auxiliary data structures. The simulation system issues are identified and quantified through prototyping using the Berkeley Topography Utilities (BTU) system which integrates SAMPLE-3D, SIMPL, and the IBM Geometry Engine. This system, which consists of 41,000 lines of C++ code, organizes geometric utilities and services as Primitive, Auxiliary, and Aggregate in a hierarchical server interface recommended for future TCAD systems. Through use of Aggregate level operations, performance tests and simulation experiments can be written in about 250 to 350 lines of C++ code. The system prototype is available for use with other servers as a whole or in parts.

IC topography simulation typically requires advancing 10,000 surface facets through 300 time steps. Major topological changes can occur in the device profile, such as opening of tunnels, and pinch off of voids. Geometry servers, such as solid modelers, offer the rigor to cope with topological changes. On the other hand, the BTU system prototype shows that considerable code augmentation (about 10,000 lines of C++) is necessary to translate between the solid modeling viewpoint and the IC topography viewpoint.

To provide reasonable performance, four essential geometry server constructs are shown to be necessary: 1) Explicit representation of connectivity information; 2) Facet sorting by location for efficient surface collision detection; 3) Ray and facet sorting for efficient line-of-sight tests; and 4) Localized deformation algorithms for incrementally moving faces. Standardized performance tests are specified vis-a-vis these constructs, and can reveal areas where geometry server design tradeoffs interact poorly with simulation needs. For example, in simulating boundary deformation, IBM Geometry Engine merge operations may repeatedly duplicate unperturbed connectivity links for robustness. By assemblying 1,600 block staircases, it can be shown that 10 seconds are required to merge 1 block or 40 blocks at a time.

A new data organization scheme called monotone decomposition is introduced to increase the granularity of the geometry representation, and to focus the robustness and power of geometry servers where they are most needed. Monotone decomposition groups large numbers of locally connected facets with similar orientation. Using this method, it is estimated that IBM Geometry Engine merge operations can simulate void formation in a 1,000 triangle surface in about 1 minute.

This thesis also introduces a new TCAD functionality called IC topography propagation trace-back, which allows users to start with a faulty topographical feature, and trace back its process flow and layout dependencies. Due to temporary masking layers, auxiliary data structures are needed to force the propagation of dependencies down to topographical features. This thesis also recommends an improved approach to process history tagging in which solid model attribution services are extended to support trace-back.

[illegible]

Professor Andrew R. Neureuther (Chair)

ACKNOWLEDGEMENTS

First, I would like to thank Professor Neureuther, who has patiently helped me make the transition from an undergraduate student to a software system specialist. His enthusiasm and open-mindededness allowed me to build up my skills in computer science and numerical analysis. Throughout my thesis research, he tried to teach me the art of clear technical presentation. His re-phrasings somehow always said what I tried to say in 1/5 the length and 10x the clarity. I only hope that I had learned enough of his skills to survive on my own.

Thanks are due to Professors Sequin and Sethian for reading this thesis. In reading several thesis drafts, Professor Sequin helped me look at my research from a solid modeling perspective. Professor Sethian exposed me to the rigor and vast potential of his level set method. I would also like to thank Professors Oldham and Hald for serving on my qualifying exam committee, and giving me valuable advice during and after the exam.

I had the good fortune of working with many bright past and present members of the Berkeley TCAD group (in choronological order): Alex Wong, Ed Scheckler, Derek Lee, John Helmsen, and John Sefler. I am also very grateful for the opportunity to collaborate with several outstanding undergraduate programmers (in chronological order): Andrej Gabara, Calvin Cheng, Terence Chin, Ravi Gunturi, Rex Winterbottom, Clark Jen, and Owen ("Pope") Carmichael. They taught me more about programming and system administration than they probably realize. They also perpetually renewed my confidence in the success and superiority of Cal's undergraduate education (modulo our football team).

This thesis would have been impossible without the help of many colleagues at IBM Research, especially those who actively participated in the TCAD Frameworks project (in alphabetical order by last name): Wally Dietrich, Michael Karasick, Derek Lieber, Lee Nackman, and V.T. Rajan. Special thanks are due to Dr. Karasick, who patiently introduced me to solid modeling and object-oriented design, and actively participated as a thesis committee member. Thanks are also due to those who worked on the 2D shock-tracking etching simulation program (in alphabetical order by last name): Manoj Dalvie, Rida Farouki, Satoshi Hamaguchi, and Vijay Srinivasan. I especially wish to thank Dr. Hamaguchi, who not only explained the details of his numerical simulator, but also furthered my understanding of the Japanese culture. Finally, I am grateful to the IBM management for letting me use the IBM Geometry Engine, and funding my entire graduate education (in conjunction with the California MICRO Program and SEMATECH/SRC).

At IBM Yorktown, several good friends made my stays so enjoyable that I decided to return after graduation. (Ironicially, many of them have left IBM since my previous stay.) I would like to thank Stacey Berlow, for not laughing too hard at all my strange adventures; Goodwin Chin, for his innumerous advice in the past few years; Charles Chu, for being my fraternity big brother, even after graduation; Rich Ferguson, for sharing his experiences as a former Andy student; Anil Kaul, for putting up with me for a whole year; and Ling-Yi and Leslie Liu, for many delicious dinners and entertaining conversations.

I would like to thank my immediate and extended families for their unrelenting support throughout my education. This thesis is dedicated to them. Thanks are also due to Catherine, who is the love of my life, and has patiently waited for me to begin our life together.

Last but not least, I thank our Heavenly Father. All through this incredible journey, He gave me the assurance that He always believed in me.

Dedicated to

My parents, for bringing me to this world and to this country,

My sister Lily, for putting up with a pigheaded brother,

My cousin Eric, for putting up with a workaholic roommate,

My cousin Paula, for being my other sister,

My younger cousins, for reminding me what's important in life,

My maternal uncles and aunts, for acting as surrogate parents,

My grandmother, for being my friend, and

God Almighty, who guiding all of our lives.

Table of Contents

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1. MOTIVATION

Many research opportunties arise from the disparity between the popular vision of what a process TCAD system should do, and the reality of available code, particularly with respect to performance issues in specific applications such as 3D IC topography simulation. The popular vision is that users would experience seamless integration of TCAD tools for a sequence of process steps [1]-[9], developers would be able to write new process simulators with minimal coding effort [8]-[11], and specialists in various computational sciences ranging from computational geometry to computational fluid mechanics would provide robust and efficient algorithms. While considerable progress has been made on the integration front through SWR prototyping and commercial systems development, only limited experimentation has been carried out for centralizing geometry services to support 3D simulation development [1][3][7][8][12][13]. Since 3D geometric algorithms are difficult and costly to develop, the centralization of geometry services clearly warrants more careful consideration.

This thesis initiates research in a new field of centralizing geometry services for 3D IC topography simulation, and makes contributions along several research fronts. The thesis explores TCAD system organization for centralizing geometry services. Issues involved in the development, performance testing, and use of centralized geometry servers are investigated. In many cases, the issues are identified and quantified through use of a prototype system based on linking geometry servers through a hierarchically organized interface.

Three-dimensional IC topography simulation poses many technical challenges for building centralized geometry servers. First, the variety of physical mechanisms which must be modeled is quite large. These mechanisms include (but are not limited to): Surface reflection (e.g.[14],[15]), Surface charging (e.g. [16]), Surface diffusion (e.g. [17],[18]), Surface reaction (e.g. [19]), and, Profile evolution (e.g. 2D:[14][16][19]-[26]; 3D:[13][15][27]-[34]). To support these mechanisms, geometry server data structures and algorithms need to be applicable in many physical situations.

Secondly, simulation of 3D profile evolution creates many geometric situations that seriously challenge geometry server robustness. For example, a well-known nemesis for geometry server robustness is the topological change as an advancing surface self-intersects and leaves behind a sealed void. This occurs in simulating low pressure chemical vapor deposition (LPCVD) over a deep trench with a narrow opening. In this case, a robust geometry server need to be able to represent this geometric situation, and break the self-intersecting surface into a valid deposition front and the void boundary.

Finally, geometry server performance in 3D IC topography simulation is strongly dependent on whether the server can exploit the special geometrical nature of IC topographies, and IC etching and deposition processes. Simulated profile time-evolution typically requires 10,000 surface faces to be advanced through about 300 time steps. To support efficient simulation of profile evolution, geometry servers need to implement spatial data structures to focus on global topological changes, and localized deformation algorithms to incrementally change facet positions and orientations. To efficiently compute surface visibility, geometry servers should exploit the fact that large numbers of locally connected surface facets have similar orientations, and that the surface undergoes incremental change between simulation time steps.

Despite the large number of physical mechanisms, the geometrical effects of these mechanisms can be conveniently modeled by initially implementing four geometric services: 1) Connectivity Services, 2) Face-Face Intersection Services, 3) Ray-Face Intersection Services, and 4) Deformation Services. For a geometry server to be effective, it needs to robustly and efficiently implement at least these four services. Achieving good performance in today's the state-of-the-art in geometry server technology is dominated by run time effects of providing these four services. As centralized geometry servers become more mature, new physical mechanisms and performance requirements will likely drive the demand to centralize other services.

This thesis considers three sources of centralized geometry services for 3D IC topography simulation. First, there are computer graphics packages, which initially appear to be an ideal source of general-purpose visibility services. However, most commercial computer graphics packages implement screen-space algorithms, such as z-buffer algorithms, which avoid line-of-sight visibility tests by drawing over objects. On the other hand, object-space computer graphics packages, such as radiosity methods, are mostly research prototypes, and are often hard coded with overly simplistic physical models for light reflection. In short, current computer graphics technology, which is one of the possible sources of centralized geometry services, is not readily applicable to IC topography simulation. (For a more comprehensive survey on computer graphics principles, see [35].)

By comparison, boundary representation solid modelers are a more promising source of general-purpose geometry services. Solid modeling was first introduced to store and manipulate large mechanical assembly. Therefore, most commercial solid modelers can represent and manipulate complex topologies inherent in 3D IC topographies, such as multiple material volumes and voids. Solid modelers typically provide point location tests, for checking point containment by material volumes, and boolean set operations, for detecting and resolving solid object collisions. In using solid modelers in 3D IC topography simulation, the main concerns are: 1) Stress on the geometric algorithms due to the large number of facets needed for physical detail (i.e. moving from 100 to 10,000 facets); 2) The amount of time to perform certain tasks on the internal server constructs; and 3) The extensive coding required to develop interface layers that map IC topography simulation geometrical operations to geometry services.

A third source of geometry services is surface-based 3D IC topography simulators. Special-purpose geometry servers can be built by consolidating efficient simulation algorithms for surface advancement, loop removal, and surface visibility. However, this approach requires working towards, rather than starting from, a common data representation. It also attempts to retroactively install robustness, rather than implicitly inherit robustness from general-purpose constructs.

Eventually, it is highly desirable to create a class of general-purpose geometry servers that implement robust and efficient algorithms for boundary deformation and surface visibility. Toward this end, an ideal TCAD system organization would be one in which a continuum of flexible choices could be made between robust general-purpose solid modeling operations, and high performance special-purpose geometric algorithms. Such a system could be used to investigate issues involved in the development, performance testing, and use of centralized geometry servers for 3D IC topography simulation.

1.2. DISSERTATION OVERVIEW

This thesis begins in Chapter 2 with a survey of geometry support functionality in 17 TCAD systems. The survey traces the history of three levels of geometry support, 1) Solid boundary generation, 2) Data mapping, and 3) Centralized geometry services. It also shows that centralization of geometry services is a logical next step in the evolution of TCAD systems. In an unusual ordering, the research presentation here begins in Chapter 3, with a discussion of a recommended hierarchical organizational structure, and the exploratory prototype system used to conduct the investigation.

A hierarchical server interface based on input data granularity of geometrical operations is recommended to manage the large number of geometric utilities and services introduced by geometry servers. The purpose for hierarchically organizing geometric utilities and services is to share codes that support data mapping, server extensions, and simulation experiments and applications. The principal test vehicle for exploring TCAD organizational issues is then introduced as the Berkeley Topography Utilities (BTU) system. The BTU system uses the recommended hierarchical interface approach to integrate Berkeley topography simulators with the IBM Geometry Engine. It is a rather extensive object-oriented (C++) system that both provides the hierarchical interface, and wraps the simulators and the IBM Geometry Engine.

Chapter 4 links the performance of geometrical operations in 3D IC topography simulation to four essential geometry server constructs. These constructs can be implemented using conventional connectivity and spatial data structures, and special-purpose boolean set operations. It is shown that the implementation of these constructs constitutes a necessary but not sufficient condition for efficient 3D IC topography simulation.

Since essential geometry server constructs are shown to be necessary for efficient 3D simulation, they may be used to screen potential servers. In Chapter 5, the essential constructs are used to examine performance aspects of a prototypical modern solid modeler, the IBM Geometry Engine. This modeler provides extensive solid model connectivity services, and robust boolean set operations. Several constructs which may lead to poor asymtotic performance behavior in an otherwise efficiently implemented geometrical operation are identified.

Chapter 6 defines standardized performance tests, which are designed to mimic the stress placed on geometry servers during 3D IC topography simulation. Since standardized performance tests take into account the nature of geometrical operations, they are an indispensable system tool for characterizing the run time consequences of theoretical performance bounds. Standardized performance tests can screen out false performance bottlenecks often predicted from simpler asymtotic performance estimates. They can also reveal areas where geometry server design tradeoffs interact poorly with IC topography simulation needs.

Chapter 7 introduces monotone decomposition as an auxiliary data organization scheme to provide large grain surface decomposition. In IC topography simulation, large numbers of locally connected facets often have similar orientations. By bin sorting locally connected facets with similar orientations, monotone decomposition can easily partition a simulated surface into large grain monotone patches. Using monotone decomposition, a surface advance with global intra-surface collisions can be broken into a few well-behaved monotone patch advances. This can efficiently focus the power and robustness of merge operations in solid modelers to the intra-surface collisions where it is most needed in IC topography simulation.

Chapter 8 introduces IC topography propagation trace-back as a new TCAD functionality. This functionality allows users to start with a faulty topographical feature, and trace back the process steps and layout masks that might have caused it. Due to the presence of temporary masking layers, such as resist layers, auxiliary data structures are needed to force the propagation of process flow and layout dependencies down to topographical features. Chapter 8 describes two auxiliary data structures that semantically extend solid model attribution services to support IC topography propagation trace-back. Improvements on attributions propagation features in solid modelers are also recommended.

The summary in Chapter 9 gives an overview of what was investigated, the results, and a perspective on their implications for IC topography simulation. Section 9.2 describes the current status and features in the BTU system, and may be of interest to TCAD developers in whole or in part. Future research in the development, performance testing, and use of centralized geometry servers for next generation 3D IC topography simulation is also suggested.

REFERENCES FOR CHAPTER 1

[1] G.M. Koppelman, M.A. Wesley, OYSTER: a study of integrated circuits as three-dimensional structures. IBM Journal of Research and Development, March 1983, vol.27, (no.2):149-63.

[2] D.C. Cole, E.M. Buturla, S.S. Furkay, K. Varahramyan, and others. The use of simulation in semiconductor technology development. Solid-State Electronics, June 1990, vol.33, (no.6):591-623.

[3] P. Lamb, C. Hegarty, N. Hitschfeld, W. Fichtner, Generating solid models for VLSI process and device simulation. Proceedings of 1992 IEEE Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA, 31 May-1 June 1992. pp. 175-80.

[4] R.H. Wang, A. Gabara, A.R. Neureuther, BTU-Berkeley Topography Utilities for linking topography and impurity profile simulations. Proceedings of 1992 IEEE Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA, 31 May-1 June 1992. pp. 225-30.

[5] D.M.H. Walker, C.S. Kellen, D.M. Svoboda, A.J. Strojwas, The CDB/HCDB semiconductor wafer representation server. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1993, vol.12, (no.2):283-95.

[6] F. Fasching, W. Tuppa, S. Selberherr, VISTA-the data level. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 1994, vol.13, (no.1):72-81.

[7] D. Yang, Mesh generation and information model for device simulation. Ph.D. Thesis, Stanford University, June 1994.

[8] P. Lloyd, C.C. McAndrew, M.J. McLennan, S.R. Nassif, and others. Technology CAD at AT&T. Microelectronics Journal, March 1995, vol.26, (no.2-3):79-97.

[9] Giles, M.D.; Boning, D.S.; Chin, G.R.; Dietrich, W.C., Jr.; and others. "Semiconductor wafer representation for TCAD," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 1994, vol.13, (no.1):82-95.

[10] Z.H. Sahul, K.C. Wang, Z-K Hsiau, E.W. McKenna, R.W. Dutton. Heterogeneous process simulation tool integration. IEEE Transactions on Semicondutor Manufacturing.

[11] Minchang Liang; Law, M.E. An object-oriented approach to device simulation-FLOODS. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Oct. 1994, vol.13, (no.10):1235-40.

[12] R.H. Wang, M.S. Karasick, and A.R. Neureuther, Computational evaluation of three-dimensional topography process simulation components, Proceedings of International Workshop on VLSI Process and Device Modeling (VPAD), Kyoto, Japan, May 1993, pp. 95-96.

[13] S. Tazawa, F.A. Leon, G.D. Anderson, T. Abe, and others. 3-D topography simulation of via holes using generalized solid modeling. Proceedings of IEEE International Electron Devices Meeting, San Francisco, 13-16 Dec. 1992. p. 173-6.

[14] J.P. McVittie, J.C. Rey, A.J. Bariya, M.M. IslamRaja, and others. SPEEDIE: a profile

simulator for etching deposition. Proceedings of the SPIE - The International Society for Optical Engineering, 1991, vol.1392:126-38.

[15] Hung Liao, T.S. Cale, Three-dimensional simulation of an isolation trench refill process. Thin Solid Films, 15 Dec. 1993, vol.236, (no.1-2):352-8.

[16] J.C. Arnold, H.H. Sawin, Charging of pattern features during plasma etching. Proceedings of the Symposia on Patterning Science and Technology II. Interconnection and Contact Metallization for ULSI. Phoenix, AZ, USA, 13-17 Oct. 1991. p. 186-94.

[17] J. Pelka, K.P. Muller, H. Mader, Simulation of dry etch processes by COMPOSITE. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1988, vol.7, (no.2):154-9.

[18] Pelka, J. Three-dimensional simulation of ion-enhanced dry-etch processes. Microelectronic Engineering, Sept. 1991, vol.14, (no.3-4):269-81.

[19] K. Harafuji, A. Misaka, H. Nakagawa, M. Kubota, and others. Profile predictions in dry-etching by a new surface reaction model. Proceedings of IEEE International Electron Devices Meeting, San Francisco, CA, USA, 13-16 Dec. 1992. p. 169-72.

[20] W.G. Oldham, S.N. Nandgaonkar, A.R. Neureuther, M. O'Toole, M. A general simulator for VLSI lithography and etching processes. I. Application to projection lithography. IEEE Transactions on Electron Devices, April 1979, vol.ED-26, (no.4):717-22.

[21] W.G. Oldham, A.R. Neureuther, J. L. Reynolds, S.N. Nandgaonkar, and others. A general simulator for VLSI lithography and etching processes. II. Application to deposition and

etching. IEEE Transactions on Electron Devices, Aug. 1980, vol.ED-27, (no.8):1455-9.

[22] I.V. Katardjiev, Simulation of surface evolution during ion bombardment. Journal of Vacuum Science & Technology A July-Aug. 1988, vol.6, (no.4):2434-42.

[23] D.S. Ross, Ion etching: an application of the mathematical theory of hyperbolic conservation laws. Journal of the Electrochemical Society, May 1988, vol.135, (no.5):1235-40.

[24] Thurgate, T. Segment-based etch algorithm and modeling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1991, vol.10, (no.9):1101-9.

[25] S. Hamaguchi, M. Dalvie, R.T. Farouki, S. Sethuraman, A shock-tracking algorithm for surface evolution under reactive-ion etching. Journal of Applied Physics, 15 Oct. 1993, vol.74, (no.8):5172-84.

[26] D. Adalsteinsson, J.A. Sethian, A level set approach to a unified model for etching, deposition, and lithography. I. Algorithms and two-dimensional simulations. Journal of Computational Physics, Aug. 1995, vol.120, (no.1):128-44.

[27] F. Jones, J. Paraszczak, RD3D (computer simulation of resist development in three dimensions). IEEE Transactions on Electron Devices, Dec. 1981, vol.ED-28, (no.12):1544-52.

[28] K.K.H. Toh, A.R. Neureuther, E.W. Scheckler, Algorithms for simulation of three-dimensional etching. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, May 1994, vol.13, (no.5):616-24.

[29] E.W. Scheckler, A.R. Neureuther, Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1994, vol.13, (no.2):219-30.

[30] C.H. Sequin, Computer simulation of anisotropic crystal etching. Sensors and Actuators A (Physical), Sept. 1992, vol.A34, (no.3):225-41.

[31] K. Brakke, The Surface Evolver. Experimental Mathematics, 1992, vol. 1, (no. 2):141-165.

[32] E. Strasser, S. Selberherr, Algorithms and models for cellular based topography simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1995, vol.14, (no.9):1104-14.

[33] J.J. Helmsen, M. Yeung, D. Lee, A.R. Neureuther, SAMPLE-3D benchmarks including high NA and thin film effects. Proceedings of the SPIE - The International Society for Optical Engineering, 1994, vol.2197:478-88.

[34] D. Adalsteinsson, J.A. Sethian, A level set approach to a unified model for etching, deposition, and lithography. II. Three-dimensional simulations. To appear Journal of Computational Physics, 1995.

[35] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Edition, Addison-Wesley Publishing Co., 1987.

CHAPTER 2

GEOMETRY SUPPORT IN TCAD SYSTEMS

2.1. INTRODUCTION

This chapter surveys current TCAD systems with respect to three levels of geometry support. As this chapter will show, current TCAD systems lack the kinds of organizational structures and centralized geometry services needed to implement efficient and semantically rich 3D IC topography simulation. In general, due to significant technical difficulties involved in developing robust and efficient 3D geometric utilities, TCAD systems may need to rely on several centralized geometry servers to provide reliable 3D geometry support. Currently, few TCAD systems which need these services provide organizational structures necessary to efficiently interoperate multiple geometry servers. Moreover, most centralized geometry servers which provide these services are designed around classical principles limited to supporting the construction and manipulation of static solid boundaries. These principles must be extended to support frequent and incremental surface movement in IC topography simulation.

A TCAD system can potentially provide three levels of centralized geometry support for different types of simulation applications. For electrical or mechanical simulation, a basic TCAD system should enable device topography construction and **solid boundary generation**. For integrated process flow simulation, a more advanced TCAD system can provide **data mapping** between radically dissimilar geometric representations, such as surface meshes used in IC topography simulation, and volume meshes used in IC dopant profile simulation. Finally, for particular classes of simulation applications, the most advanced TCAD systems can provide **centralized geometry services** to simplify physical model implementation. For instance, for 3D IC topography simulation, a TCAD system could provide a set of sophisticated and efficiently integrated 3D geometric utilities for computing boundary deformation and surface visibility.

Tables 2.1a-c ([1]-[30][36]) list seventeen current TCAD systems, the system developers, and the geometry support provided in these systems. These TCAD systems represent a wide range of application areas and organizational structures. In terms of application areas, these systems cover from dopant profile and topography simulation, to device simulation and capacitance extraction. In terms of organizational structure, while some of the systems are monolithic simulation programs, such as SAMPLE-3D [6], many others are TCAD systems that integrate several tools, such as AT&T's Integrated TCAD system [1] and the committee-designed SWR [36]. A more comprehensive review of current TCAD system capabilities can be found in an anthology of TCAD system review papers, such as [31].

This chapter critiques current TCAD systems in terms of these three geometry support levels: **1) Solid Boundary Generation,, 2) Data Mapping**, and **3) Centralized Geometry Services**. In Tables 2.1a-c, implemented geometry support levels are denoted by **X**'s in the appropriate columns. In each TCAD system, there are two criteria that determine its geometry support levels. The first criterion is the services that can be provided by the underlying geometry server. The second criterion is how well TCAD tools can use these services. Based on these criteria, it will be shown that centralized geometry servers currently lack the constructs needed to support efficient 3D IC topography simulation. Moreover, this chapter shows that current TCAD systems lack organizational structures needed to conveniently and efficiently interoperate centralized geometry servers.

Table 2.1a

Geometry Support in TCAD Systems

TCAD System	Organization	Geometry Servers	Solid Boundaries for Electrical Simulation	Data Mapping for Integrated Process Flow Simulation	Centralized Services for Physical Modeling
Integrated TCAD System [1]	AT&T	BSP Tree Modeler [2], PROPHET [3]	X	X	X
SIMPL [4]	UC Berkeley	old BTU [5]		X	
SAMPLE 3D [6]	UC Berkeley	Deloop [7], Solid Extract [8]	X		X
PROSE [9]	UC Berkeley	old BTU [5], BPIF [10]		X	
SWR [37]	CFI / Sematech	Geometry Engine [19]	X	X	X

Table 2.1b

Geometry Support in TCAD Systems

TCAD System	Organization	Geometry Servers	Solid Boundaries for Electrical Simulation	Data Mapping for Integrated Process Flow Simulation	Centralized Services for Physical Modeling
PREDITOR [11]	CMU	CDB [11]	X	X	
OMEGA Tools [12]	ETH Zurich (now ISE)	Echidna [13]	X		
FLOODS [14] / FLOOPS	University of Florida			X	X
OYSTER [15]	IBM	GDP [16,17]	X		
VATS [18]	IBM	VATS DB [18]	X	X	X
FOXI [19] / FIERCE [18]	IBM	Geometry Engine [19]	X		

Table 2.1c

Geometry Support in TCAD Systems

TCAD System	Organization	Geometry Servers	Solid Bound- aries for Electrical Simulation	Data Mapping for Integrated Process Flow Simulation	Centralized Services for Physical Modeling
EASE [20]	Intel	TIF [21]		X	
THUNDER- BIRD [22]	Intel + NTT	ACIS [23], GWB [24]	X		X
MEMCAD [25]	MIT	Proprietary [26]	X		
CAMINO Interface [27]	Stanford	ACIS [23]	X		
Forest [28]	Stanford	HDF–V Set [29]		X	X
VISTA [30]	TU Vienna	VPIF [30]		X	

2.2. SOLID BOUNDARY GENERATION

Solid modeling operations have long been used in TCAD systems to generate realistic device boundaries for volume mesh generation and 3D device simulation. In 1983, IBM first developed the OYSTER [15] system, which creates 3D device topographies as input to FIELDAY [33], a finite element device simulation program. OYSTER used the GDP [16][17] solid modeler to create 3D IC topography components with rounded corners by sweeping out layout mask openings and oxidation profiles rotationally and translationally. Using GDP's **merge** operation (i.e. a boolean set operation), OYSTER stitched together theses topography components into a device structure. Similar techniques for device topography construction were subsequently adopted and refined by researchers at ETH Zurich [12] to simulate CMOS and bipolar devices using the Echidna [13] solid modeler, AT&T Allentown [1] to simulate SRAM cells using a BSP Tree [3] solid modeler, and Stanford [27] to simulate SRAM cells using the ACIS [23] solid modeler.

TCAD systems have also used layout mask extrusion and solid geometry stitching to generate 3D input structures for capacitance extraction and microelectromechanical (MEM) simulation. In 1990, IBM reported the FOXI [19] system, which uses the IBM Geometry Engine [19] to stitch solid primitives, such as boxes, cones, and spheres, into complex device structures, such as a DRAM cell, as input to FIERCE [18], a finite element capacitance extraction program. Shortly after, in 1992, MIT reported on its MEMCAD [25] system, which includes a proprietary solid modeler [26] for constructing 3D MEM structures by layout mask extrusion and boolean set operations.

Recent simulation studies have demonstrated the need for using rigorous 3D IC topography simulation to generate input structures for electrical analysis. For example, in [8], two pairs of polysilicon elbows were generated using layout mask extrusion and SAMPLE-3D etching simulation. The authors compared mutual capacitance values calculated by FASTCAP [35], MIT's fast multipole capacitance extraction program, on both the extruded and SAMPLE-3D simulated 3D elbow structures. Due to geometrical differences such as etch bias and sidewall curvatures, the mutual capacitance values of the two structures can differ by as much as 30% [8].

With respect to most TCAD systems, solid boundary generation can be considered a mature technology. The third column of Tables 2.1a-c summarizes the current status of supporting solid boundary generation in TCAD systems. As listed in these tables, 12 out of 17 TCAD systems are capable of performing 3D solid boundary generation. For a new TCAD system, this wealth of experience suggests that solid boundary generation would be a natural starting point for building up 3D geometry support.

2.3. DATA MAPPING

Successful 2D data mapping implementations [5][28] suggest that solid modelers will be indispensable for integrating 3D surface-representation-based IC topography simulations. To incorporate surface-representation-based topography simulation results, a **stitch-back** utility is needed to add new layers to wafer geometry, or to clip the wafer geometry against etched surfaces. In 2D, stitch-back involves only polygon and string intersections. Therefore, the 2D stitch-back function can be easily developed from scratch. On the other hand, the 3D stitch-back function requires robust and efficient solid and surface intersection computations. Hence, it would be best to implement this function by extending a 3D geometry server's solid or surface intersection operations.

While supporting 3D data mapping may seem to be a worthwhile near term goal, the significant cost associated with developing 3D computational algorithms suggests moving on to more long term goals. In fact, the few TCAD systems that currently support limited 3D data mapping have achieved this as a result of centralizing computational services. The fourth column of Tables 2.1a-c summarizes the current status of supporting 2D and 3D data mapping in TCAD systems. At present, only 2 out of 17 TCAD systems, AT&T's Integrated TCAD System [1] and IBM's VATS [18] system, support limited 3D data mapping. In both cases, 3D data mapping is supported as the result of centralizing volume mesh services for 3D thermal process and device simulation programs.

2.4. CENTRALIZED GEOMETRY SERVICES

As far back as the early 1980s, IBM had implicitly centralized 3D field services by sharing FORTRAN finite element simulation modules between FIELDAY [33], FEDSS [34], and FIERCE [18]. In 1990, AT&T introduced the PROPHET [3] field server, which is the first object-oriented implementation of centralized services for 2D and 3D PDE solution and volume mesh generation. Since 1990, PROPHET C++ object classes have been succesfully embedded in AT&T's Integrated TCAD System to implement production-line proven simulation models for 2D and 3D oxidation, dopant diffusion, and device characterization [1]. Recently, university researchers have followed with Florida's FLOODS/FLOOPS [14] system, and Stanford's Forest [28] system. However, these systems are designed to support university research on cutting-edge physical models for 2D oxidation, dopant diffusion, and device simulation.

Current TCAD systems that support centralized computation services primarily target thermal process and device simulation, and contain organizational structures that do not interoperate servers. The fifth column of Tables 2.1a-c summarizes the current status of supporting 2D and 3D centralized computational services for physical model development. As listed in Tables 2.1a-c, 5 out of 17 systems centralize volume mesh services for 2D or 3D thermal process and device simulation, but only 3 out of 17 systems provide centralize geometry services for 3D topography simulation. Morever, to avoid data mapping between geometrically dissimilar 3D data representations, most of these TCAD systems contain organizational structures that restrict the numbers and the types of geometry servers. Since robust and efficient 3D geometric algorithms are difficult to develop, an ideal TCAD system should implement an organizational structure which more freely interoperates geometry servers.

REFERENCES FOR CHAPTER 2

[1] P. Lloyd, C.C. McAndrew, M.J. McLennan, S.R. Nassif, and others. Technology CAD at AT&T. Microelectronics Journal, March 1995, vol.26, (no.2-3):79-97.

[2] B.F. Naylor, Interactive solid geometry via partitioning trees. Proceedings. Graphics Interface '92, Vancouver, BC, Canada, 11-15 May 1992. p. 11-18.

[3] Pinto, M.R.; Coughran, W.M., Jr.; Rafferty, C.S., Jr.; Smith, R.K.; and others. Device simulation for silicon ULSI. In: *Computational Electronics: Semiconductor Transport and Device Simulation*. Edited by: K. Hess, et al. Kluwer Academic Publishers, 1991. p. 3-13.

[4] D. Lee, Applying TCAD to Emerging Technologies, MS Thesis, UC Berkeley, UCB/ERL M95/38, May 20 1995.

[5] R.H. Wang, A. Gabara, A.R. Neureuther, BTU-Berkeley Topography Utilities for linking topography and impurity profile simulations. Proceedings of 1992 IEEE Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA, 31 May-1 June 1992. pp. 225-30.

[6] E.W. Scheckler, A.R. Neureuther, Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1994, vol.13, (no.2):219-30.

[7] J.J. Helmsen, E.W. Scheckler, A.R. Neureuther, C.H. Sequin, An efficient loop detection and removal algorithm for 3D surface-based lithography simulation. Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA,

31 May-1 June 1992, p. 3-8.

[8] J. Sefler, 3D Surface Modeling Utilities for use in TCAD, MS Thesis, UC Berkeley, October 28, 1995.

[9] A.S. Wong, D.M. Newmark, J.B. Rolfson, R.J. Whiting, and others. Investigating phase-shifting mask layout issues using a CAD toolkit. IEEE International Electron Devices Meeting 1991. Washington, DC, USA, 8-11 Dec. 1991, p 705-8.

[10] A.S. Wong, A.R. Neurcuther, The intertool profile interchange format: a technology CAD environment approach (semiconductor technology). IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1991, vol.10, (no.9):1157-62.

[11] D.M.H. Walker, C.S. Kellen, D.M. Svoboda, A.J. Strojwas, The CDB/HCDB semiconductor wafer representation server. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1993, vol.12, (no.2):283-95.

[12] P. Lamb, C. Hegarty, N. Hitschfeld, W. Fichtner, Generating solid models for VLSI process and device simulation. Proceedings of 1992 IEEE Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA, 31 May-1 June 1992. pp. 175-80.

[13] A. Paoluzzi, M. Ramella, A. Santarelli, Boolean algebra over linear polyhedra. Computer Aided Design, Oct. 1989, vol.21, (no.8):474-84.

[14] Minchang Liang; Law, M.E. An object-oriented approach to device simulation-FLOODS. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,

Oct. 1994, vol.13, (no.10):1235-40.

[15] G.M. Koppelman, M.A. Wesley, OYSTER: a study of integrated circuits as three-dimensional structures. IBM Journal of Research and Development, March 1983, vol.27, (no.2):149-63.

[16] M.A. Wesley, L.I. Lieberman, M.A. Lavin, D.D. Grossman, and others. A geometric modeling system for automated mechanical assembly. IBM Journal of Research and Development, Jan. 1980, vol.24, (no.1):64-74.

[17] R.N. Wolfe, M.A. Wesley, J.C. Kyle, Jr., F. Gracer, and others. Solid modeling for production design. IBM Journal of Research and Development, May 1987, vol.31, (no.3):277-95.

[18] D.C. Cole, E.M. Buturla, S.S. Furkay, K. Varahramyan, and others. The use of simulation in semiconductor technology development. Solid-State Electronics, June 1990, vol.33, (no.6):591-623.

[19] M. Karasick, D. Lieber, "Schemata for Interrogating Solid Boundaries," ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[20] J. Mar, K. Bhargavan, S.G. Duvall, R. Firestone, and others. EASE-an application-based CAD system for process design. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Nov. 1987, vol.CAD-6, (no.6):1032-8.

[21] S.G. Duvall, An interchange format for process and device simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, July 1988, vol.7, (no.7):741-54.

[22] S. Tazawa, F.A. Leon, G.D. Anderson, T. Abe, and others. 3-D topography simulation of via holes using generalized solid modeling. IEEE International Electron Devices Meeting 1992. Technical Digest. San Francisco, CA, USA, 13-16 Dec. 1992. p. 173-6.

[23] Spatial Technology Inc. 2425 55th Street, Bldg. A, Boulder, Colorado 80301-5704 USA, Telephone 303-449-0649, Email info@spatial.com, WWW http://www.spatial.com/spatial.

[24] M. Mantyla, *Introduction to solid modeling*, Rockville, MD, USA: Computer Science Press, 1988.

[25] S.D. Senturia, R.M. Harris, B.P. Johnson, S. Kim, and others. A computer-aided design system for microelectromechanical systems (MEMCAD). Journal of Microelectromechanical Systems, March 1992, vol.1, (no.1):3-13.

[26] R.M. Harris, F. Maseeh, S.D. Senturia, Automatic generation of a 3-D solid model of a microfabricated structure. IEEE Solid-State Sensor and Actuator Workshop. Technical Digest. 1990. p. 36-41.

[27] D. Yang, Mesh generation and information model for device simulation. Ph.D. Thesis, Stanford University, June 1994.

[28] Z.H. Sahul, K.C. Wang, Z-K Hsiau, E.W. McKenna, R.W. Dutton, "Heterogeneous Process Simulation Tool Integration," IEEE Trans. on Semiconductor Mfg, to appear.

[29] L. Bishop, U. Ravaioli, P. Fu, D. Yergeau, Z. Sahul, D. Yang, R.W. Dutton, and R. Goossens, HDF-VSet file format for visualization of mesh-based simulation data, Technical Report, Stanford University, October 1992.

[30] F. Fasching, W. Tuppa, S. Selberherr, VISTA-the data level. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 1994, vol.13, (no.1):72-81.

[31] F. Fasching, S. Halama, S. Selberherr (Editors), *Technology CAD Systems*, Springer-Verlag/Wien, 1993.

[33] E.M. Buturla, P.E. Cottrell, B.M. Grossman, K.A. Salsburg, Finite-element analysis of semiconductor devices: the FIELDAY program. IBM Journal of Research and Development, July 1981, vol.25, (no.4):218-31.

[34] K.A. Salsburg, H.H. Hansen, FEDSS-finite-element diffusion-simulation system. IEEE Transactions on Electron Devices, Sept. 1983, vol.ED-30, (no.9):1004-11.

[35] K. Nabors, J. White, FastCap: a multipole accelerated 3-D capacitance extraction program. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Nov. 1991, vol.10, (no.11):1447-59.

[36] M.D. Giles, D.S. Boning, G.R. Chin, W.C. Dietrich, Jr.; and others. Semiconductor wafer representation for TCAD. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan. 1994, vol.13, (no.1):82-95.

CHAPTER 3
BERKELEY TOPOGRAPHY UTILITIES

3.1. INTRODUCTION

An ideal TCAD system should offer a continuum of choices between general-purpose services and special-purpose services for IC process and device simulation. This chapter recommends a hierarchical server interface based on input data granularity of geometrical operations to manage the large number of geometric utilities and services introduced by geometry servers. The purpose for hierarchically organizing geometric utilities and services is to share codes that support data mapping, server extensions, and simulation experiments and applications. The principal test vehicle for exploring TCAD organizational issues is then introduced as the Berkeley Topography Utilities (BTU) system [1]. The BTU system uses the recommended hierarchical interface approach to integrate Berkeley topography simulators with the IBM Geometry Engine. It is a rather extensive object-oriented (C++) system that both provides the hierarchical interface, and wraps the simulators and the IBM Geometry Engine.

The BTU system organizational structure consists of five layers which are believed to be essential to future geometry service based TCAD systems. The five BTU system layers are: **1) Centralized Geometry Servers, 2) Primitive Server Interface, 3) Auxiliary Server Interface, 4) Aggregate Server Interface**, and **5) Simulation Support Utilities**. The Centralized Geometry Server layer currently supports both general-purpose servers, such as the IBM Geometry Engine [2], and special-purpose servers, such as SAMPLE-3D [3] and SIMPL System 6 [4]. Collectively, the Primitive, Auxiliary, Aggregate Server Interface layers are known as **BTU Hierarchical Interfaces**. These three layers uniformly integrate and hierarchically organize geometry services under an **object-oriented** application procedural interface. The Simulation Support Utilities layer complements geometry services with simulation specific utilities for simulation task management and visualization. Section 3.3 describes a system integration scheme that presents an integrated and hierarchically organized geometry server interface, and maintains geometry server software boundaries.

Section 3.4 highlights BTU experimental applications that interoperate the IBM Geometry Engine with Hamaguchi's 2D shock tracking topography simulation program [5], SAMPLE-3D, and SIMPL. To demonstrate application development convenience and server interoperability achieved through **BTU Hierarchical Interfaces**, Section 3.4 briefly describes solid structures and surface meshes constructed by these tests and experiments. Similar tests and experiments will be presented throughout Chapters 6, 7, and 8.

Section 3.5 discusses the use of the BTU system organization as an infrastructure for implementing 3D IC topography simulation application. Section 3.5 first analyzes SAMPLE-3D source code distribution with respect to BTU system layers. This analysis assesses the coding effort required to implement a 3D simulation application based on a fully implemented BTU system. Section 3.5 further discusses how the BTU system organization could enhance the robustness and maintenance of 3D simulation applications.

3.2. BTU SYSTEM LAYERS

The **Berkeley Topography Utilities (BTU)** system is an object-oriented (C++) five-layer system that integrates general-purpose geometry servers, such as the IBM Geometry Engine, with special-purpose geometry servers, such as SAMPLE-3D and SIMPL System 6. This section introduces the five BTU system layers: **1) Centralized Geometry Servers, 2) Primitive Server Interface, 3) Auxiliary Server Interface, 4) Aggregate Server Interface,** and **5) Simulation Support Utilities.** Figure 3.1 illustrates BTU system layers and applications. In Figure 3.1, BTU system layers are denoted using rectangular boxes, and BTU applications are represented using bubbles.

As shown at the bottom of Figure 3.1, **Centralized geometry servers**, the main subject of study in this thesis, lie at the foundation of the BTU system. Due to significant performance disparities between server types, the BTU system explicitly distinguishes general-purpose servers, such as the IBM Geometry Engine, from special-purpose servers for IC topography simulation, such as SAMPLE-3D and SIMPL System 6. General-purpose servers can provide robust geometrical operations on solid boundary representations. On the other hand, special-purpose servers can perform similar geometrical operations more efficiently by representing and manipulating only surface elements. Eventually, these servers could converge into a new generation of general-purpose servers that may efficiently perform the kinds of geometrical operations used in IC topography simulation.

As shown in the left of Figure 3.1, the BTU system divides the application procedural interface to centralized geometry servers into three layers: The Primitive, Auxiliary, and Aggregate Server Interfaces. Collectively, these interfaces are known as **BTU Hierarchical Interfaces.** These three layers uniformly integrate and hierarchically organize geometry services under an object-oriented application procedural interface. Fundamental concepts in object-oriented programming such as **encapsulation**, **inheritance**, and **polymorphism**, are used in BTU Hierarchical Interfaces to wrap geometry server functionalities for TCAD applications, and to facilitate the development and application of auxiliary data structures. Detailed discussions on object-oriented programming can be found in [6].

The **Primitive Server Interface** layer wraps small-grain geometry services, such as **surface connectivity** and **boolean set operations**. This layer contains **geometry server wrapper objects**, or object classes that **encapsulates** geometry server data structures and algorithms. Table 3.1 summarizes the object classes and the amount of code contained in the three server interface layers. As shown in the bottom of Table 3.1, Primitive Server Interface objects include wrappers for **surface connectivity**, and utilities for **polygon triangulation**. Other Primitive Server Interface objects include wrappers for CPU intensive geometrical operations such as **boolean set operations** and **line-of-sight visibility tests**. In terms of code size, the Primitive Server Interface layer dominates the BTU Hierarchical Interfaces prototype at about 30,750 lines of C++ code.

The **Auxiliary Server Interface** layer implements auxiliary data structures for efficient use and semantics extension of geometry services. As shown in the middle of Table 3.1, auxiliary data structures currently supported in the BTU system include **monotone decomposition**, **process history tagging**, and **topography propagation trace-back**. Detailed descriptions of these data structures will be given later in Chapters 7 and 8. To facilitate the implementation and application of auxiliary data structures, such as **monotone decomposition**, BTU Hierarchical Interfaces apply well-established object-oriented programming concepts. As will be described in Chapter 7, monotone decomposition partitions a surface mesh into a few large-grain monotone patches. In implementing monotone decomposition, **inheritance** was used to share codes between the **MonotonePatch** and **SurfaceMesh** object classes. Inheritance was also used to share codes among a family of 3D monotone decomposition methods. **Polymorphism** enabled aggregate geometric utilities, such as boundary deformation, to perform the same geometric computations on monotone patches generated by different monotone decomposition methods. As a result, the **MonotonePatch** object class, two 3D monotone decomposition methods, and geometric computations on 3D monotone decomposition, were implemented in a total of 1,200 lines of C++ code.

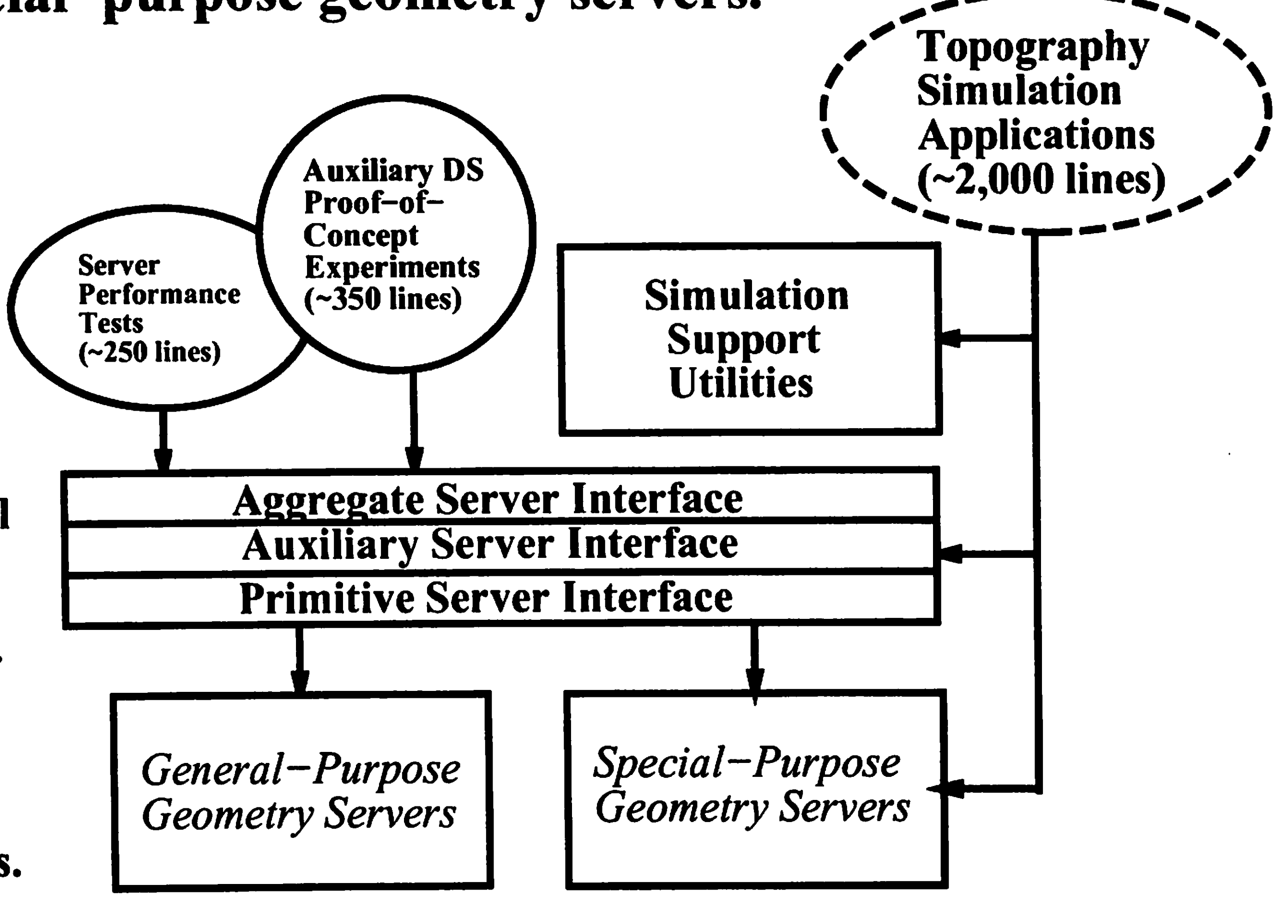
The **Aggregate Server Interface** layer implements large grain geometrical operations, such as boundary deformation and source visibility. An aggregate geometric utility can be implemented by directly wrapping large-grain geometry services, such as SAMPLE-3D's **source visibility** operation. Alternatively, an aggregate geometric utility can be implemented by combining Auxiliary Server Interface objects, such as **monotone decomposition**, with Primitive Server Interface objects, such as **line-of-sight visibility tests.**

As shown in the right top corner of Figure 3.1, **Simulation support utilities** are textual and graphical utilities specific to IC topography simulation. **Pre-processing simulation support utilities**, capture simulation input from the end user, and execute the simulation inner loop. These utilties include graphical user-interface (GUI) utilities, input deck parsers, and simulation task management utilities. **Post-processing simulation support utilities** include computer graphics utilities for visualization of IC topography simulation results, and geometrical analysis of the visualized solids and surfaces.

In terms of code size, the BTU system organization can provide excellent application development leverage, although the cost of adding geometry servers can be significant. Figure 3.1 compares the codes sizes of BTU applications versus BTU system layers. As will be discussed in Section 3.4, using BTU Hierarchical Interfaces, experimental applications can be written in about 250 to 350 lines of C++ code. As will be analyzed in Section 3.5, using utilities and services provided by a fully implemented BTU system, rigorous 3D IC topography simulation applications could be written in about 2,000 lines of C++ code. On the other hand, as listed in the lower right corner of Figure 3.1, the BTU Hierarchical Interfaces prototype currently contains about 41,000 lines of C++ code to interface 4 geometry servers. In other words, adding a geometry server to the BTU system could require about 10,000 lines of C++ code.

Figure 3.1

Berkeley Topography Utilities (BTU) is a five-layer, object-oriented system for integrating general-purpose and special-purpose geometry servers.


Topography Simulation Applications (~2,000 lines)
Server Performance Tests (~250 lines)
Auxiliary DS Proof-of-Concept Experiments (~350 lines)
Simulation Support Utilities
Aggregate Server Interface
Auxiliary Server Interface
Primitive Server Interface
General-Purpose Geometry Servers
Special-Purpose Geometry Servers

BTU Hierarchical Interface Prototype: 41,000 lines.

Typical Geometry Server: ~40,000 lines.

Table 3.1

Primitive Interface dominates in terms of code size, Auxiliary Interface dominates in terms of sophistication.

Interface Layer	Object Classes	Code Size
Aggregate Server Interface	2.5D and 3D Boundary Deformation, 3D Source Visibility.	5,330 lines of C++.
Auxiliary Server Interface	2D and 3D Monotone Decomposition, 2D Process History Tagging and Trace–Back.	4,920 lines of C++.
Primitive Server Interface	2D and 3D Solid and Surface Connectivity, 3D Sweep, Polygon Triangulation, 3D Tiled Primitives.	30,750 lines of C++.

3.3. BTU SYSTEM INTEGRATION

The BTU system integration strategy aims at providing a convenient development environment for both TCAD developers and geometry server developers. To comply with different development responsibilities, the system integration strategy need to simultaneously provide a convenient TCAD system organization and preserve well-established geometry server software boundaries. For example, Figure 3.2 illustrates the BTU system directory tree. As shown in the right of Figure 3.2, BTU Hierarchical Interfaces wraps disjoint geometry server components, into an integrated conglomeration of hierarchical interface objects. On the other hand, as shown in the left of Figure 3.2, geometry servers in the BTU system preserve their individual software boundaries by maintaining separate directory trees.

In the BTU system directory tree, Unix file links are used to simultaneously provide a convenient development environment and maintain geometry server software boundaries. In Figure 3.2, file links are plotted as dashed arrows. As depicted in the bottom of Figure 3.2, files links go from hierarchical interface directories to geometry server directories. These links are used to present geometry server wrapper objects as BTU Hierarchical Interface objects. For example, as shown in the left of Figure 3.2, the **SAMPLE-3D** directory may contain three **SAMPLE-3D** wrapper objects: **Surface**, **Octree**, and **Source Visibility**. Using file links from the **SAMPLE-3D** sub-directories to the appropriate **Interfaces** sub-directories, a TCAD application could include **Surface** as a Primitive Server Interface object, **Octree** as an Auxiliary Server Inverface object, and **Source Visibility** as an Aggregate Server Interface object.

Files Links enables Geometry Server Developers to maintain software ownership within the BTU system organization.

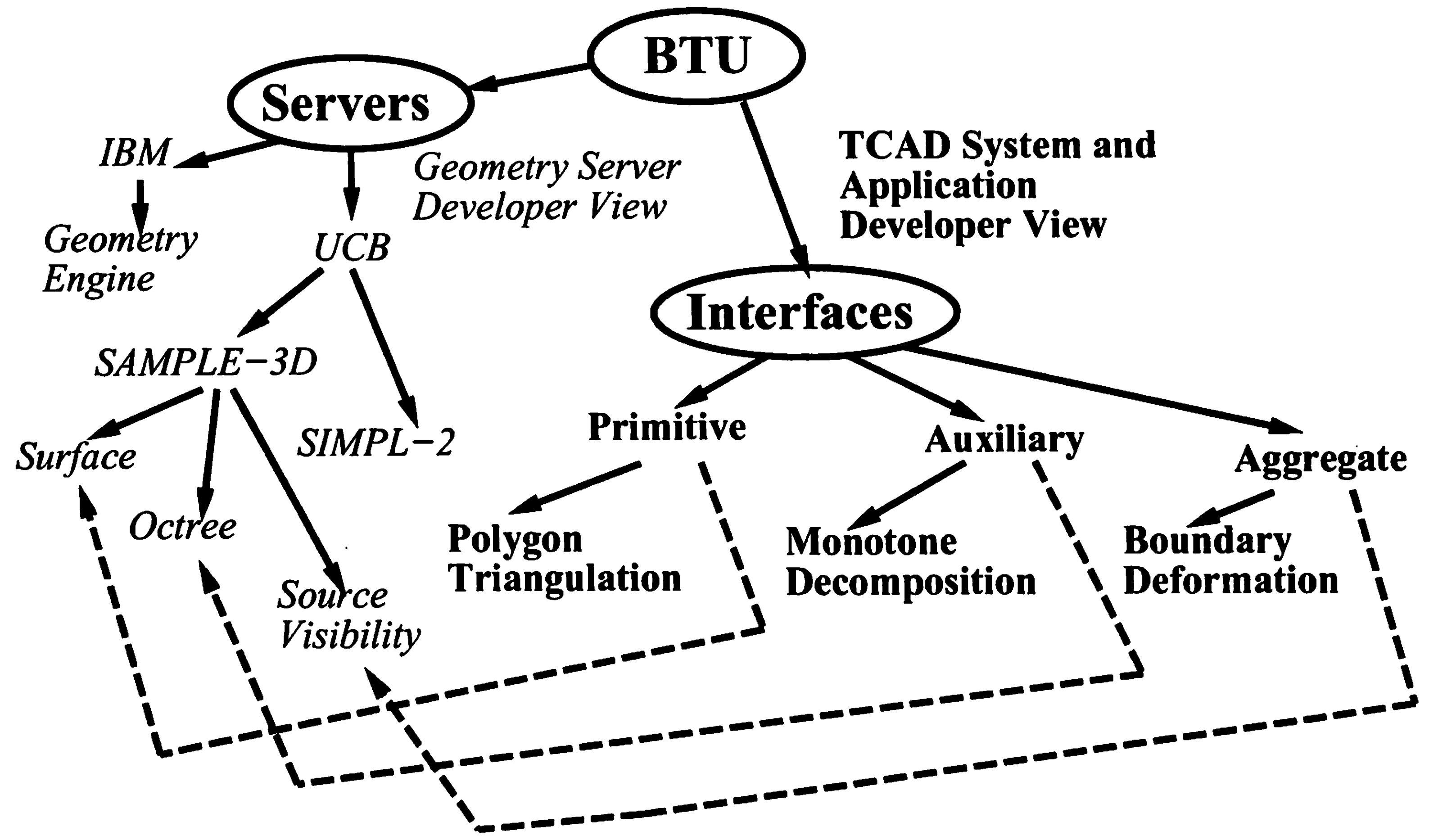


Figure 3.2

3.4. BTU EXPERIMENTAL APPLICATIONS

This section highlights epxerimental applications that demonstrate the application development convenience and server interoperability achievable through the BTU system. The experimental results to be presented in this section were obtained by interoperating the IBM Geometry Engine with special-purpose geometry servers, such as Hamaguchi's 2D shock tracking topography simulation program, SAMPLE-3D, and SIMPL. Using BTU Hierarchical Interfaces, experimental applications to be described here were written in about 250 to 350 lines of C++ code.

The first examples of BTU experimental applications are IBM Geometry Engine **standardized performance tests** [7]. As will be described in Chapter 6, these tests characterized the performance of the IBM Geometry Engine using geometrical operations in 3D IC topography simulation. BTU system provides geometric utilities to construct solid structures for IBM Geometry Engine performance testing. The left of Figure 3.3 illustrates several solid structures constructed by these utilities. The bottom left corner of Figure 3.3 depicts a two-holes initial structure created using tiled planar layers, and two inverted cone sections. The top left corner of Figure 3.3 depicts a triangulated vertical deposition volume generated from the top surface of the two-holes initial structure. To the right of the vertical deposition volume is a staircase that was constructed by merging cubes. This structure was used to chracterize the performance of repeated boolean set operations.

In the **2.5D Isotropic Deposition Experiment**, the BTU system was used to interoperate the IBM Geometry Engine with Hamaguchi's 2D shock tracking topography simulation program. As will be described in Chapter 7, this experiment involved simulating a 0.3 um isotropic deposition on a 1 um deep 2.5D key hole trench, with a 0.25 um opening at the top of the trench. The right of Figure 3.3 depicts the resulting solid structure generated by this experiment. In this figure, the front face of the solid structure shows the surfaces generated by the 2D shock tracking solver at three time steps leading up to time (T) = 1 second. At T = 1 second, the isotropic deposition created a void in the topography. This void is efficiently detected and resolved using 2D monotone decomposition, and two IBM Geometry Engine **merge** operations.

To enable the use of IBM Geometry Engine **merge** operations in simulating 3D boundary deformation, the BTU system implemented 3D **monotone decomposition** auxiliary data structures. Monotone decomposition groups large numbers of similar orientation facets, into a few large grain monotone surface patches [1]. As will be demonstrated in Chapter 7, 3D monotone decomposition was used to efficiently interoperate IBM Geometry Engine **merge** operations with SAMPLE-3D **surface advancement** operations. Figure 3.4 illustrates a monotone decomposition of a 3D key hole trench surface with 896 triangles. The left of Figure 3.4 shows the IBM Geometry Engine solid structure used to extract the key hole trench surface. On the right of Figure 3.4, the key hole trench surfach is decomposed into 9 monotone patches. As will be discussed in Chapter 7, this decomposition can reduce the number of IBM Geometry Engine **merge** operations from 895 down to 8.

To analyze the causes of topography propagation effects, the BTU system implemented a (material) volume-based **process history tagging** auxiliary data structure for attaching and propagating process flow and layout dependencies on IC topographical features. As will be discussed in Chapter 8, a 2.5D process history tagging data structure was implemented to tag SIMPL process step and layout mask id's on IBM Geometry Engine solid structures. Chapter 8 will also describe a new TCAD function known as **topography propagation trace-back.** This function traverses process history tags, and reports the process flow and layout dependencies of IC topographical features.

In this section, the **Metal Stringer Trace-Back Experiment** is used to demonstrate interoperation of the IBM Geometry Engine with SIMPL. This experiment began by using SIMPL to simulate an IC topography that contains a metal stringer propagated from an underlying polysilicon line. After each SIMPL topography process step, the resulting deformation volume was extruded and updated in the IBM Geometry Engine. Trace-backs were then performed on the metal stringer and the poly line.

The SIMPL process flow simulated by the **Metal Stringer Trace-Back Experiment** is listed in Figure 3.5. Figure 3.6 shows the SIMPL layout and final cross section obtained from this process flow simulation. As listed in Figure 3.5, the major sequences in this process flow are: **Poly 1** deposition and etching (thickness = 1 um, mask = POLY, Steps 1 through 7); **Oxide 1** deposition (thickness = 1 um, Step 8); **Metal 1** deposition and etch-back (thickness = 1 um, Steps 9 through 11).

As one would expect, topography propagation trace-back of the **Metal 1** stringer and the **Poly 1** line showed that the lithography and etching of **Poly 1** line strongly influenced the shape of the **Metal 1** stringer. Figure 3.7 depicts the 3D topography simulated by the IBM Geometry Engine, and compares the dependencies reported for the metal stringer (listed on the right), and that for the poly line (listed on the left). From the reported dependencies, it can be seen that the **Metal 1** stringer not only depended on the metal deposition and etch-back (Steps 9 and 11), but also depended on the poly line deposition, lithography, and etching (Steps 1, 2, 4, 6, and POLY mask).

IBM Geometry Engine Solid Structures Generated by BTU Applications

3D Geometry Server Performance Testing (~250 lines C++ / Test)

2.5D Key Hole Trench Deposition Simulation (~350 lines C++ / Simulator)

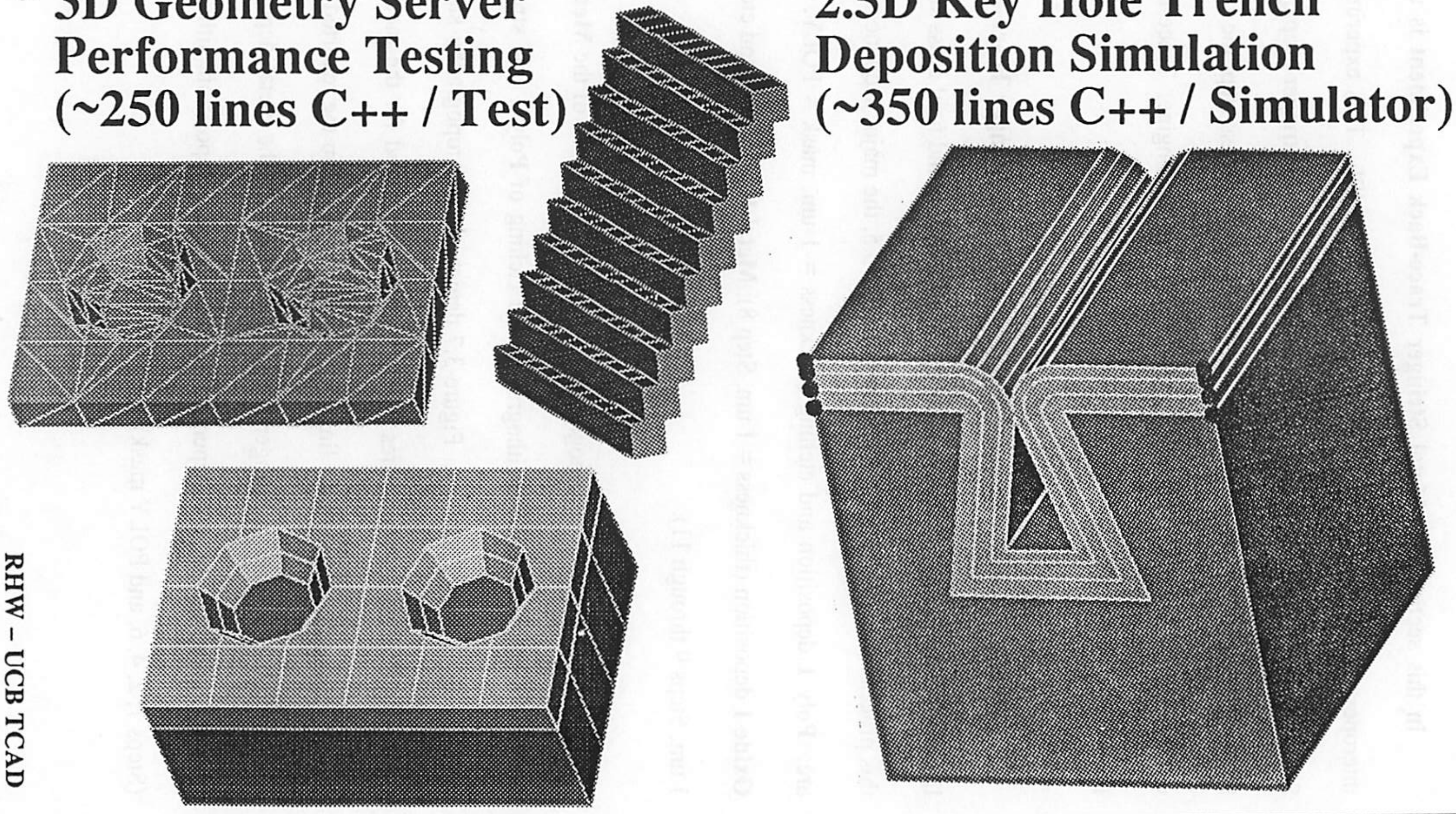
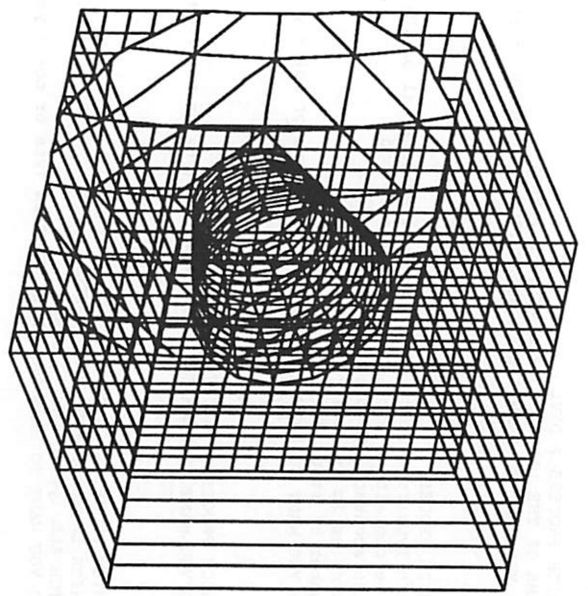


Figure 3.3

RHW – UCB TCAD

Monotone Decomposition of 3D Key Hole Trench

3D Key Hole Trench (896 Triangles)



Monotone Patches (9 Patches)

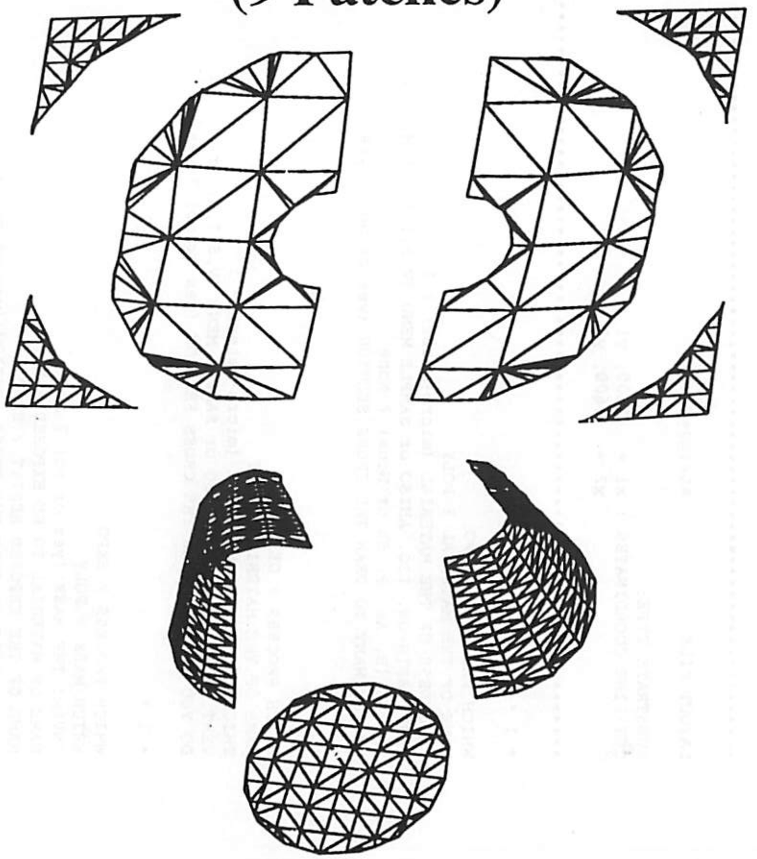


Figure 3.4

RHW – UCB TCAD

Metal Stringer Test SIMPL Process Flow

```
****************************************************************************

LAYOUT FILE :          stringer2.cif

SUBSTRATE TYPE:
CUT-LINE COORDINATES : x1 =  -1600, y1 =    -47
                       x2 =   1600, y2 =    -47

****************************************************************************

* 1 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? POLY
THICKNESS OF THE MATERIAL (micro-meter) ? 1
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? V
DOPING (B, As, P, Sb or None) ? None
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 2 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? RST
THICKNESS OF THE MATERIAL (micro-meter) ? 1
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? S
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 3 *

WHICH PROCESS ? EXPO
WHICH MASK ? POLY
INVERT THE MASK (yes or no) ? no
NAME OF MATERIAL TO BE EXPOSED ? RST
NAME OF THE EXPOSED RESIST ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 4 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 5 *

WHICH PROCESS ? ETCN
Etch Type:Isotropic, or Iso with Directional (1 or 10) ? 10
File containing etch rates ? poly.etch.mod
Etch accuracy (0:worst to 10:best) ? 10
Timestep in seconds ? 11
Number of steps ? 1
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 6 *

WHICH PROCESS ? ETCU
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 7 *

WHICH PROCESS ? ETCH
WHICH LAYER DO YOU WANT TO ETCH ? RST
ETCH ALL (yes or no) ? yes
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes
```

Figure 3.5

Metal Stringer Test SIMPL Process Flow (Continued)

```
* 8 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? OXID
THICKNESS OF THE MATERIAL (micro-meter) ? 1
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? I
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 9 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? METL
THICKNESS OF THE MATERIAL (micro-meter) ? 1
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? I
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 10 *

WHICH PROCESS ? ETCN
Etch Type:Isotropic, or Iso with Directional (1 or 10) ? 10
File containing etch rates ? metal.etch.mod
Etch accuracy (0:worst to 10:best) ? 10
Timestep in seconds ? 11
Number of steps ? 1
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 11 *

WHICH PROCESS ? ETCU
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? END
```

Figure 3.5
(Continued)

Metal Stringer Test Layout and Cross Section

(SIMPL System 6 Simulation)

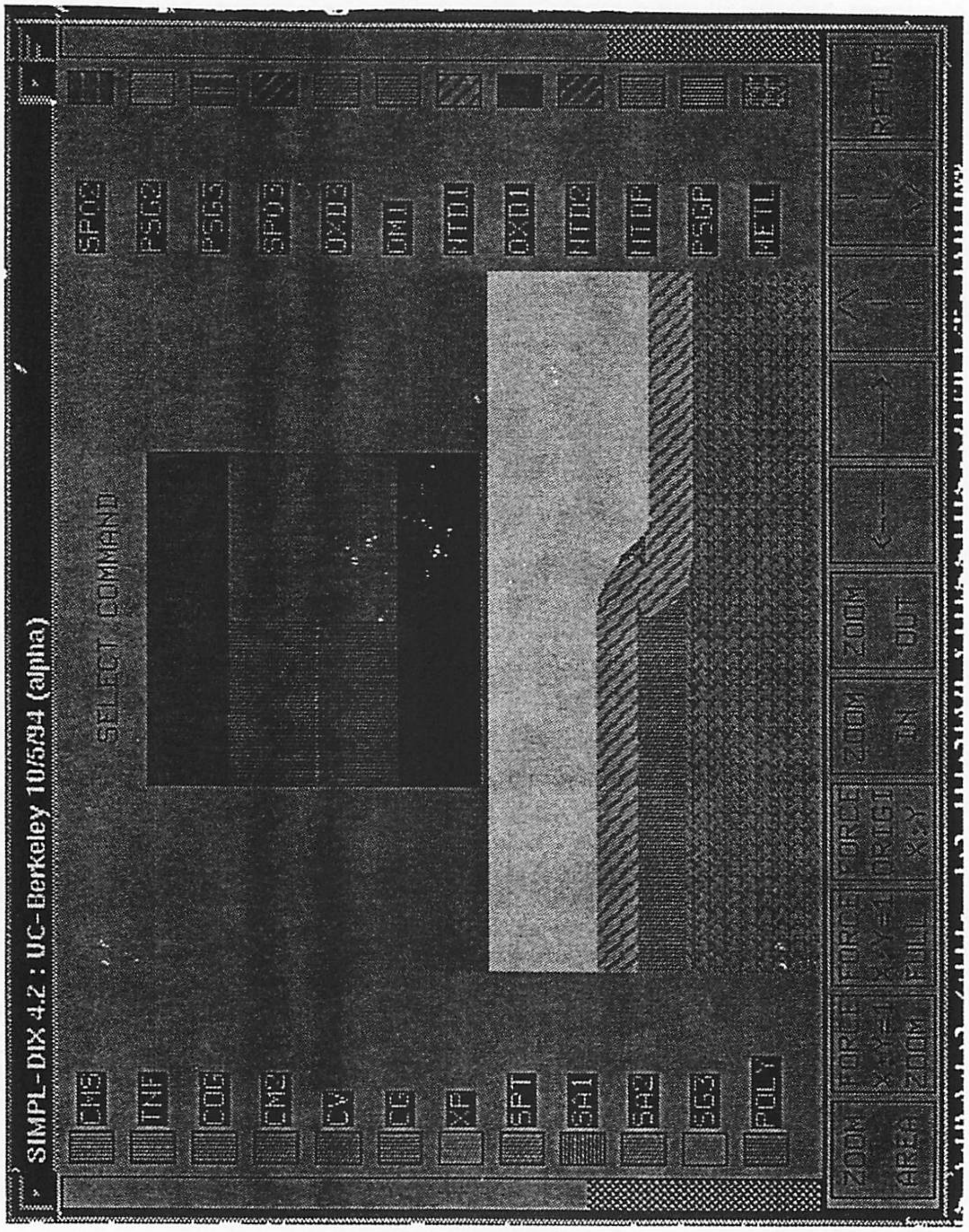
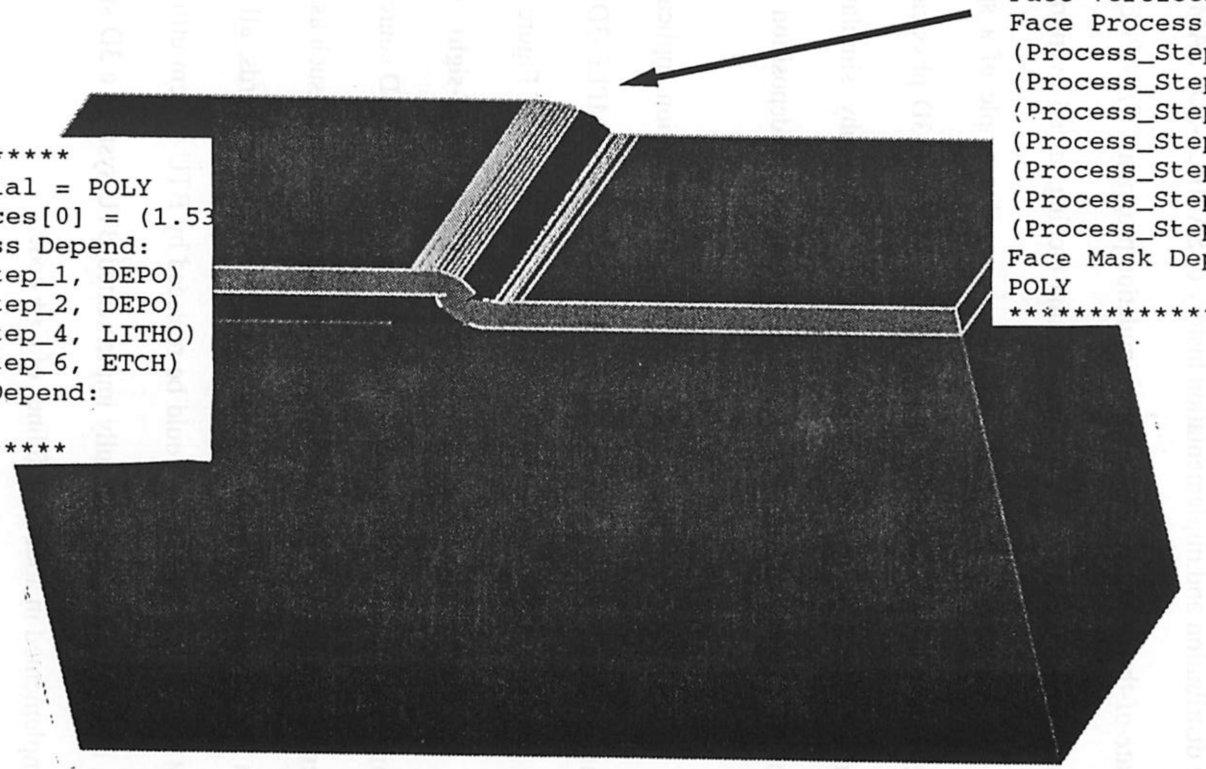


Figure 3.6

RHW – UCB TCAD

Shape of the metal stringer is depedent on the poly line litho–etch mask and process sequence.



```
***************
Face Material = METL
Face Vertices[0] = (1.59
Face Process Depend:
(Process_Step_9, DEPO)
(Process_Step_11, ETCH)
(Process_Step_8, DEPO)
(Process_Step_2, DEPO)
(Process_Step_4, LITHO)
(Process_Step_6, ETCH)
(Process_Step_1, DEPO)
Face Mask Depend:
POLY
***************
```

```
***************
Face Material = POLY
Face Vertices[0] = (1.53
Face Process Depend:
(Process_Step_1, DEPO)
(Process_Step_2, DEPO)
(Process_Step_4, LITHO)
(Process_Step_6, ETCH)
Face Mask Depend:
POLY
***************
```

Figure 3.7

RHW – UCB TCAD

3.5. BTU SIMULATION APPLICATIONS

This section evaluates the effectiveness of the BTU system organization in implementing 3D IC topography simulation applications. The evaluation involves studying the source code distribution and implementation history of SAMPLE-3D. SAMPLE-3D is representative of state-of-the-art 3D IC topography simulation programs. SAMPLE-3D implements physical models for lithography, ion milling, plasma etching, metal evaporation, sputter deposition, and chemical vapor deposition. It is also a good example of a special-purpose geometry server that undergoes constant development. SAMPLE-3D physical models and geometric algorithms have been used in other 3D IC topography simulation programs, such as EVOLVE-3D [8] and VISTA's 3D cell-based etching and deposition program [9].

Using a fully implemented BTU system, 3D simulation applications could be written in about 2,000 lines of C++ code. Figure 3.8 illustrates SAMPLE-3D source code distribution with respect to BTU system layers. As shown in the left of Figure 3.8, primitive, auxiliary, and aggregate geometrical operations, such as **line-of-sight visibility**, **octree**, **cell-decomposition**, and **deloop**, account for 73% of SAMPLE-3D source code. As shown in the lower right corner of Figure 3.8, simulation support utilites, such as simulation task management and visualization, take up another 22%. In other words, all but 1,800 lines (5%) of SAMPLE-3D's 36,000 lines could be replaced by BTU system utilities and services. Therefore, conservatively, using a fully implemented BTU system, a 3D simulation application can be implemented in about 2,000 lines of C++ code.

Besides centralization of geometry services, the BTU system organization improves over monolithic simulation programs, such as SAMPLE-3D, in terms of robustness and ease of maintenance. As suggested in the bottom of Figure 3.8, the robustness of special-purpose geometry operations, such as **deloop**, often require several (3) generations to perfect. The resulting **deloop** operation is difficult to improve because it is fraught with duplicate and extraneous code fragments. On the other hand, using the BTU system, simulation applications can acquire robustness as the BTU system incorporate new and proven servers.

In terms of maintenance, simulation applications built using a special-purpose server may undergo frequent code changes. This is because these applications are usually implemented using the server's internal data structures and algorithms. As a result, modifications in the server lead to modifications in simulation applications. BTU simulation applications can be easier to maintain because the applications interact with geometry servers through wrappers. Geometry server wrappers encapsulate the internal data structures and algorithms of geometry servers, and only change when geometry servers change their external data representations or functional behaviors. As a result, simulation applications built using geometry server wrappers are more stable.

Figure 3.8

Over 90% of SAMPLE–3D (36,000 lines) can be replaced by BTU system services and utilities.

Centralizable Geometry Services and Utilities

Aggregate Services — **10%** — Advance, Deloop, Visible Source Flux Integrate.

Auxiliary Services — **6%** — Octree, Cells.

Primitive Services — **57%** —
25% = Deloop Primitives.
– 1/3 = KToh and EWS.
– 2/3 = Octree Deloop.
19% = Meshing Primitives.
3% = Line–Of–Sight Visibility.

Physical Model Implementation (1,800 lines).

Physical Models — *5%*

Simulation Support — **22%** — Visualization, Task Management.

RHW – UCB TCAD

3.6. CONCLUSIONS

This chapter recommended a hierarchical organizational structure for future TCAD systems. The **Berkeley Topography Utilities (BTU)** system, which exemplifies this architecture, consists of: **1) Centralized Geometry Servers, 2) Primitive Server Interface, 3) Auxiliary Server Interface, 4) Aggregate Server Interface,** and **5) Simulation Support Utilities.** The BTU system hierarchically organized geometric utilities and services along their input data granularity under Primitive, Auxiliary, and Aggregate Server Interfaces. These three layers, known as **BTU Hierarchical Interfaces**, implement geometry server wrappers and auxiliary data structures that address complex issues arising from the use of centralized geometry services. Based on experience with the BTU Hierarchical Interfaces prototype, adding a geometry server to the BTU system would require about **10,000** lines of C++ code.

BTU Hierarchical Interfaces facilitate implementation and application of auxiliary data structures by using fundamental concepts in **object-oriented programming**, such as inheritance and polymorphism. As an example, Section 3.2 considered the implementation of the **monotone decomposition** auxiliary data structure. As will be discussed in Chapter 7, monotone decomposition partitions a surface mesh into a few large-grain monotone patches. In implementing monotone decomposition, **inheritance** was used to share codes between the **SurfaceMesh** and the **MonotonePatch** object classes. Inheritance was also used to share code among a family of 3D monotone decomposition methods. **Polymorphism** enabled aggregate geometric utilities, such as boundary deformation, to perform the same geometric computations on monotone patches generated by different monotone decompostion methods. As a result of using inheritance and polymorphism, the **MonotonePatch** object class, two 3D monotone decomposition methods, and geometric computations on 3D monotone decomposition, were implemented in a total of **1,200** lines of C++ code.

Section 3.3 described a BTU system integration strategy that created a convenient development environment for both TCAD application developers and geometry server developers. To preserve geometry server software boundaries, the BTU directory tree contains separate subtrees for geometry server objects and hierarchical interface objects. Unix file links going from hierarchical interface directories to geometry server directories allow geometry server objects to be presented as hierarchical interface objects.

Section 3.4 highlighted BTU experimental applications that interoperated the IBM Geometry Engine, a 2D shock tracker, SAMPLE-3D, and SIMPL. Section 3.4 presented solid structures and surface meshes constructed by these tests and experiments. These experimental results demonstrated the kinds of application development convenience and server interoperability achievable through BTU Hierarchical Interfaces. Using BTU Hierarchical Interfaces, experimental applications can be written in about **250** to **350** lines of C++ code.

Section 3.5 discussed the use of the BTU system organization to implement 3D IC topography simulation applications. First, by bin sorting SAMPLE-3D modules (**36,000** lines of C code) with respect to BTU system layers, it was shown that **95%** of SAMPLE-3D source code could be classifed as BTU system utilies and services. The remaining **5%**, or **1,800** lines of C code, implemented several rigorous physical models. Conservatively, this analysis suggested that, using geometric utilities and services provided by a fully implemented BTU system, a 3D simulation application could be written in about **2,000** lines of C++ code. Morever, the BTU system organization offers improved robustness through acquisition of new geometry servers, and ease of maintenance through **encapsulation** of geometry servers.

REFERENCES FOR CHAPTER 3

[1] R.H. Wang, and A.R. Neureuther, Efficient and Innovative Use of Three-Dimensional Geometry Services in IC Topography Simulation. International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA), Taipei, Taiwan, ROC, June 1995.

[2] M. Karasick, D. Lieber, "Schemata for Interrogating Solid Boundaries," ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[3] E.W. Scheckler, A.R. Neureuther, Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1994, vol.13, (no.2):219-30.

[4] D. Lee, Applying TCAD to Emerging Technologies, MS Thesis, UC Berkeley, UCB/ERL M95/38, May 20 1995.

[5] S. Hamaguchi, M. Dalvie, R.T. Farouki, S. Sethuraman, A shock-tracking algorithm for surface evolution under reactive-ion etching. Journal of Applied Physics, 15 Oct. 1993, vol.74, (no.8):5172-84.

[6] G. Booch, *Object-oriented analysis and design with applications*. Redwood City, CA, Benjamin/Cummings.

[7] R.H. Wang, M.S. Karasick, and A.R. Neureuther, Computational evaluation of three-dimensional topography process simulation components. International Workshop on VLSI Process and Device Modeling (VPAD), Kyoto, Japan, May 1993, pp. 95-96.

[8] Hung Liao, T.S. Cale, Threc-dimensional simulation of an isolation trench refill process. Thin Solid Films, 15 Dec. 1993, vol.236, (no.1-2):352-8.

[9] E. Strasser, S. Selberherr, Algorithms and models for cellular based topography simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1995, vol.14, (no.9):1104-14.

CHAPTER 4
ESSENTIAL GEOMETRY SERVER CONSTRUCTS FOR EFFICIENT 3D IC TOPOGRAPHY SIMULATION

4.1. INTRODUCTION

This chapter shows that the performance of centralized geometry servers, such as the IBM Geometry Enginc [1], is inherently linked to the support of **essential geometry server constructs**. Four constructs are presented: **1) Explicit Connectivity, 2) Face-Face Intersection Sorting, 3) Ray-Face Intersection Sorting,** and **4) Localized Deformation.** Table 4.1 summarizes the relationship between essential constructs and server performance. The middle columns of Table 4.1 list the server performance improvements gained through using these constructs. The rightmost column of Table 4.1 lists general design issues that arise during construct implementation. This chapter will describe geometric data structure and algorithm examples that address these design issues for efficient 3D IC topography simulation.

Section 4.2 uses geometrical operations in 3D IC topography simulation to introduce and motivate the four **essential geometry server constructs**. At every simulation time step, Explicit Connectivity is needed to efficiently support tens of thousands of surface connnectivity queries used to simulate surface diffusion and surface reaction. Face-Face Intersection Sorting is needed to perform efficient surface collision detection and surface loop removal in surface-based topography simulation. Ray-Face Intersection Sorting is needed to efficiently support tens of thousands of **point location tests** used to detect material interface collision, and millions of **line-of-sight visibility tests** used to determine surface visibility. Localized Deformation is needed to avoid extraneous duplications of geometry components and connectivity links during incremental boundary deformation.

Section 4.3 discusses **Explicit Connectivity**. Explicit Connectivity directly links between topologically connected geometry components to efficiently answer large numbers of surface connectivity queries at each simulation time step. As shown in the top row of Table 4.1, without Explicit Connectivity, *O(N)* time is required to answer each surface connectivity query. This is because linear searches are needed to find all geometry components connected to a query component. With Explicit Connectivity, all connected components at a vertex or an edge can be found by traversing a few (typically less than 10) links, and each surface connectivity query can be answered in *O(1) (constant)* time.

For **Explicit Connectivity**, the key design issue is to choose the type of geometry component to store most of the connectivity links, and the kinds of connectivity links to be stored. As examples, Section 4.3 will describe the 2D **winged-edge** data structure and the 3D **star-edge** data structure, which is a 3D variant of the**winged-edge** data structure supported by the IBM Geometry Engine.

Section 4.4 discusses **Face-Face Intersection Sorting**. Face-Face Intersection Sorting groups faces by spatial locality, such that face-face intersections are only computed between faces within the same proximity. As shown in the second row of Table 4.1, without Face-Face Intersection Sorting, $O(N^2)$ time is required to check for intersections between every face and every other face. With Face-Face Intersection Sorting, each face is only checked for intersection against the few faces close to it, and all surface self-intersections can be found in $O(N \log N)$ time for tree-based data structures, and $O(N^{\frac{3}{2}})$ time for grid-based data structures.

For **Face-Face Intersection Sorting**, the key design issue is to choose a spatial data structure for sorting faces, and the spatial data structure parameters (such as tree depths or voxel sizes) that optimize storage cost and intersection computation time. As examples, Section 4.4 will describe several 3D spatial data structures currently implemented in TCAD applications, such as **octrees, cell-decomposition, BSP trees.**

Sections 4.5 discusses **Ray-Face Intersection Sorting.** Ray-Face Intersection Sorting groups faces by spatial locality, such that ray-face intersections are only computed between faces which lie in the ray's proximity. As shown in the third row of Table 4.1, without Ray-Face Intersection Sorting, $O(N)$ time is required to intersect a ray against every surface facet or IC topography face. With Ray-Face Intersection Sorting, each ray is only checked for intersection against the few faces close to it, and all ray-face intersections in the surface or the topography can be found in $O(\log N)$ time, for tree-based data structures, and $O(N^{\frac{1}{2}})$ time, for grid-based data structures.

For **Ray-Face Intersection Sorting**, the key design issues are the same as that of Face-Face Intersection Sorting, plus the need to amortize the initialization cost of spatial data structures over large numbers of ray tests. Section 4.5 will discuss the adaptation of Face-Face Intersection Sorting data structures to simultaneously support Ray-Face Intersection Sorting.

Section 4.6 discusses **Localized Deformation. Localized Deformation** provides robust facet pushing to efficiently update facet positions and connectivity links of moving solid boundaries in IC topography simulation. As shown in the bottom row of Table 4.1, without Localized Deformation, worst case $O(N^2 \log N)$ time might be used to update solid boundaries at each simulation time step. This is because N facets could be trivially swept into triangular prisms, and merged using N **boolean set operations.** For robustness, each boolean set operation might duplicate unperturbed connectivity links to create intermediate versions of the aggregate deformation volume. In this scenario, each boolean set operation might use $O(N_1 \log N_1)$ time, where N_1 is the number of faces in the intermediate volume. Summing N_1

from 1 to N results in a worst case total **merge** time of *O(N^2 logN)*. With Localized Deformation, facet positions and connectivity links can be incrementally updated in *O(NlogN)* time.

For **Localized Deformation**, the key design issue is to find robust and efficient heuristics for updating connectivity links after localized facet pushing. As examples, Section 4.6 will describe two special-purpose **boolean set operations** suitable for 3D IC topography simulation. **Surface-based boolean set operations**, such as the **deloop** operation, can be used to remove invalid surfaces after surface facet pushing. **Localized boolean set operations** can be used to delimit and push small subsets of solid model faces, and establish connectivity links between subset faces. By using triangular prisms to delimit deformed faces, this approach can avoid duplication of unperturbed connectivity links. As a result, each localized boolean set operation can be performed in *O(n log n)* time, where n is the number of subset faces (typically less than 10). This results in a more *O(N)*-like total **merge** time of *O(N * n log n)*.

Table 4.1

Server Efficiency gained through Essential Server Constructs

Construct	Performance w/o Construct	Performance with Construct	Key Design Issue
Explicit Connectivity	O(N) time per query.	O(1) time per query.	Component to contain links; Types of links.
Face–Face Intersection Sorting	O(N^2) time per surface intersection.	O(NlogN) or O(N^1.5) time per surface intersection.	Spatial data structure type; Tree depth or Voxel size.
Ray–Face Intersection Sorting	O(N) time per ray test.	O(logN) or O(N^0.5) time per ray test.	Same as Face–Face Sorting; Amortize init–ialization cost.
Localized Deformation	O(N^2 logN) time per boundary deformation.	O(NlogN) time per boundary deformation.	Ability to push existing facets robustly.

RHW – UCB TCAD

4.2. MOTIVATION AND OVERVIEW

Geometrical operations in IC topography simulation can be implemented using a handful of geometry services. For efficient 3D process and device simulation, TCAD researchers have implemented special-purpose versions of connectivity data structures, such as **triangular surface meshes** [2][3][4][11], and **polygonal boundary representations** [10], spatial data structures, such as **cell-decomposition** [3][13], **octrees** [4][5][6], and **BSP trees** [7][8], and surface advancement algorithms, such as **recursive ray trace** [2], **facet motion** [3][10][11][12], **cell removal** [9][13], and **level set methods** [14][15]. This section groups these data structures and algorithms as four **essential geometry server constructs: 1) Explicit Connectivity, 2) Face-Face Intersection Sorting, 3) Ray-Face Intersection Sorting,** and **4) Localized Deformation.** One might expect commercial solid modeling and computer graphics packages to have implemented these constructs, and can make immediate impact on 3D IC topography simulation. However, as will be discussed in this section, most general-purpose geometry servers, such as the IBM Geometry Engine, typically do not implement **Localized Deformation**, and therefore cannot efficiently support 3D moving surface simulation.

Explicit Connectivity is needed to efficiently support tens of thousands of surface connnectivity queries used at every time step to simulate surface diffusion and surface reaction. Figure 4.1 illustrates use of connectivity services in surface diffusion and surface reaction simulation. Surface diffusion involves particle redistribution between neighboring surface vertices. As shown in the left of Figure 4.1, explicit representation of incident edges can be used to efficiently find the connected vertex neighbors at each surface vertex. Depending on the materials incident on a surface vertex, surface reaction creates equilibrium concentrations of visible particles and reacting species at the vertex. As shown in the right of Figure

4.1, explicit representation of incident faces and volumes can be used to efficiently find the materials meeting at each surface vertex. Usually, a surface vertex has 3 to 6 incident edges and facets, and 1 to 3 incident volumes. Therefore, for N vertices, simulation of surface diffusion and surface reaction spawns *O(N)* connectivity queries. For a typical simulation with 10,000 surface vertices (N = 10,000), about 50,000 connectivity queries are invoked at each time step.

Face-Face Intersection Sorting is needed to perform efficient surface collision detection and surface loop removal after every few time steps of a surface-based IC topography simulation. Figure 4.2 illustrates the use of face-face intersection services in surface collision detection and wafer geometry update. As shown in the left of Figure 4.2, surface-based IC topography simulators, such as SAMPLE-3D [2][3][4] and EVOLVE-3D [11], need geometrical utilities to detect and resolve global topological changes, such as void formation. After simulating a topography process step, as shown in the middle and the right of Figure 4.2, the aggregate deformation volume need to be stitched onto the wafer geometry. Face-face intersection services, such as **deloop** and **glue**, can be used to perform these tasks. Since surface collisions can occur at any time during the simulation, face-face intersection services need to be invoked after every few time steps. For a typical surface-based IC topography simulation, this translates to about 300 face-face intersection service calls on a surface with 10,000 vertices.

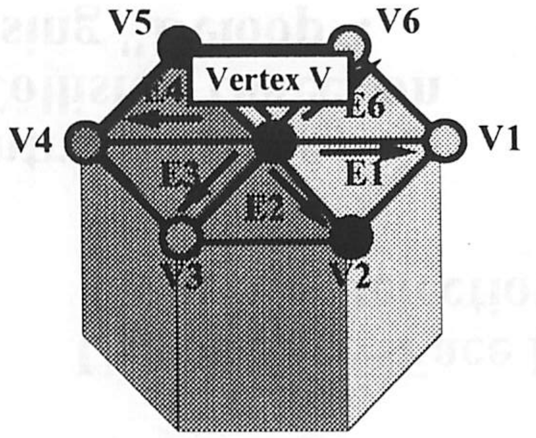
Ray-Face Intersection Sorting is needed to efficiently support tens of thousands of **point location tests** used at every time step to detect material interface collision, and millions of **line-of-sight visibility tests** used at every time step to compute surface visibility. Figure 4.3 illustrates the use of ray-face intersection services in material interface collision detection and surface visibility computation. During each time step, a surface-base etching simulation needs to check if its etch front points had crossed over a material interface. As shown in the left of Figure 4.3, a **point location test** can be used to find the material volume that contains an etch front point after its advancement. To track source particles incident on a surface during a simulation time step, the simulator needs to determine the source points that are visible to each surface vertex. As shown in the right of Figure 4.3, a **line-of-sight visibility test** can be used to determine if a source point is visible to a surface vertex. For a surface with N vertices and a source with S points, at each simulation time step, a total of O(N) **point location tests** and O(N*S) line-of-sight tests may be performed on the surface. For a typical simulation with about 10,000 surface vertices (N = 10,000) and 500 source points (S = 500), about 10,000 **point location tests** and 5 million **line-of-sight visibility tests** may be performed at each time step.

Localized Deformation is needed to avoid extraneous duplications of geometry components and connectivity links during incremental boundary deformation. Figure 4.4 illustrates the inefficiencies that can result from the lack of Localized Deformation. The left of Figure 4.4 depicts how facets can be trivially swept into small deformation volumes, and the order in which they can be merged into an aggregate deformation volume. The right of Figure 4.4 illustrates how these small volumes can be merged using **boolean set operations**. For robustness, conventional boolean set operations, such as the ones in the IBM Geometry Engine, establishes output solid connectivity links by duplicating input solid connectivity links. For the case of merging N small deformation volumes, conventional boolean set operations may incur in $O(N^2)$ extraneous duplications. As an example, the right of Figure 4.4 shows the number of extraneous duplications for each facet deformation volume. As shown in Figure 4.4, the 8 merge operations (N = 8) would incur a total of 35 (or about $\frac{N^2}{2}$ = 32) extraneous duplications.

Use of Connectivity Services in Surface Diffusion and Surface Reaction Simulation

Surface Diffusion

Incident edges { E1, ..., E6 } can be used to find connected vertex neighbors at vertex V.



Surface Reaction

Incident faces {F1, F2} can be used to find incident Volumes 1, 2. These volumes are used to find materials meeting at vertex V.

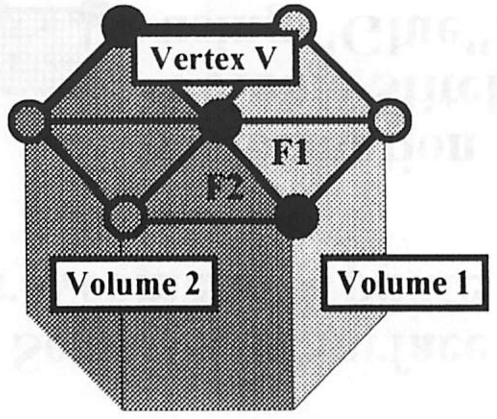


Figure 4.1

Use of Face–Face Intersection Services in Surface Collision Detection and Wafer Geometry Update

Intra–Surface Collision Detection using "Deloop".

Deformation Volume Stitching using "Glue".

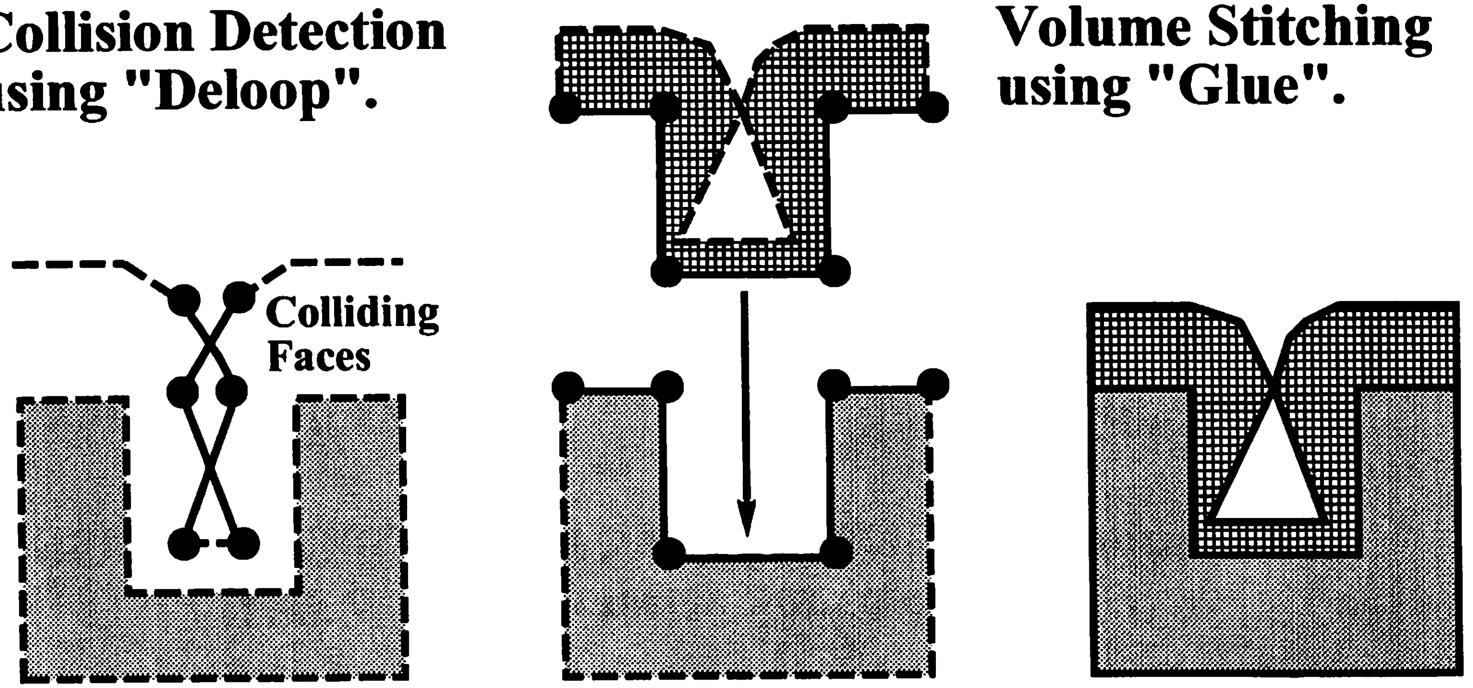


Figure 4.2

Figure 4.3

Use of Ray–Face Intersection Services in Material Interface Collision and Source Visibility

Material Interface Collision Detection

Aggregation of Point Location Test

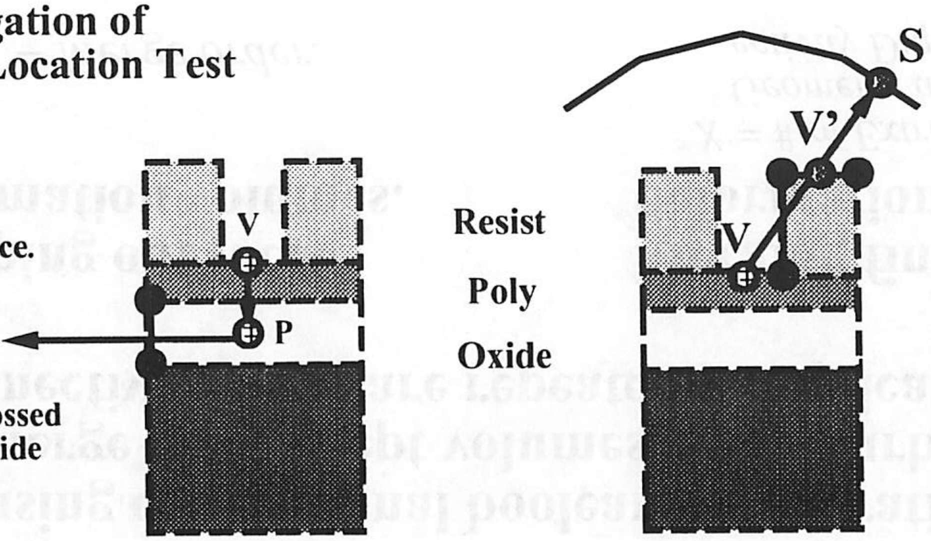
Vertex V is at Poly–Air Interface.

Point P is inside Oxide.

-> Ray (V,P) crossed over Poly–Oxide Interface.

Source Visibility Computation

Aggregation of Line–of–Sight Visibility Test



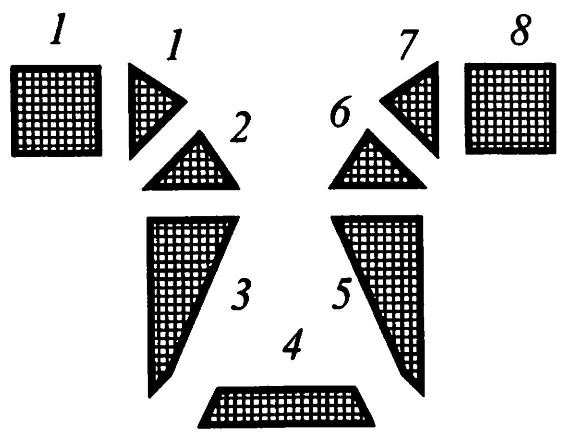
Vertex V can’t see Source Point S or Vertex V’.

Figure 4.4

In using conventional boolean set operations to merge facet swept volumes, unperturbed connectivity links are repeatedly duplicated.

Sweeping out facet deformation volumes.

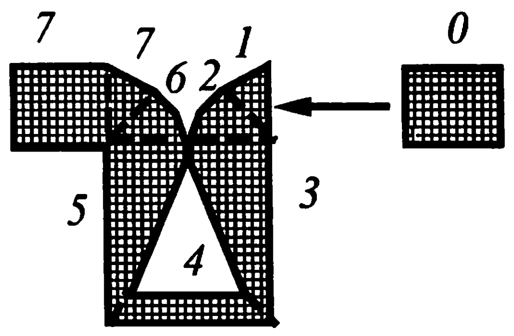
X = Merge order.



Merging final facet deformation volume.

X = # of Extraneous Geometry and Conn–ectivity Duplications.

Dashed segments have been removed.



RHW – UCB TCAD

4.3. EXPLICIT CONNECTIVITY

Explicit Connectivity directly links between topologically connected geometry components to efficiently answer large numbers of surface connectivity queries at each simulation time step. Without Explicit Connectivity, each connectivity query would require *O(N)* time to complete. For example, to find all edges incident on a vertex, every solid boundary or surface edge would have to be checked for containment of the query vertex. It can be shown that a solid or a surface with N vertices has *O(N)* edges and *O(N)* faces [16]. Therefore, without Explicit Connectivity, *O(N)* time would be required to find all edges incident on a vertex.

With **Explicit Connectivity**, each connectivity query can be answered in *O(1) (constant)* time. A common approach to implement an Explicit Connectivity data structure is to choose one type of geometry component for storing a set of connectivity links that enables *O(1)* time connectivity queries. For example, Figure 4.5 depicts the connectivity links in the 2D **winged-edge** [17] data structure. As illustrated in the left of Figure 4.5, the **winged-edge** data structure uses **edges** to store connectivity links to component vertices, incident faces, and connected edges. To facilitate connectivity queries, each vertex contains a back pointer to an incident edge, and each face contains a back pointer to a component edge. As depicted in the right of Figure 4.5, through vertex V1's back pointer to edge E1, V1's incident edges E1, E2, and E3, and incident faces F1 and F2, can be readily found by traversing E1's connectivity links. In other words, all edges and faces incident on a **winged-edge** vertex can be found in *O(1)* time.

Since Baumgart's seminal work on the 2D **winged-edge** data structure in 1972, several 3D extensions, such as the **star-edge** [18] data structure, have been developed. The **star-edge** data structure has been used to implement the IBM Geometry Engine. Figure 4.6 depicts the connectivity links in the 3D **star-edge** data structure. As illustrated in the left of Figure 4.6, the **star-edge** data structure uses **vertices** to store connectivity links to directed edges and faces. In the **star-edge** data structure, each vertex contains several **stars**. A star is used to refer to incident edges contained by the same face. For instance, as shown in the right of Figure 4.6, vertex V groups its incident edges into three stars, and the star in face F1 has 4 directed edges.

Figure 4.5

Explicit Connectivity Links in 2D Winged−Edge Data Structure

Edge store most of the connectivity links.
Vertex or face has back pointer to only one incident or component edge.

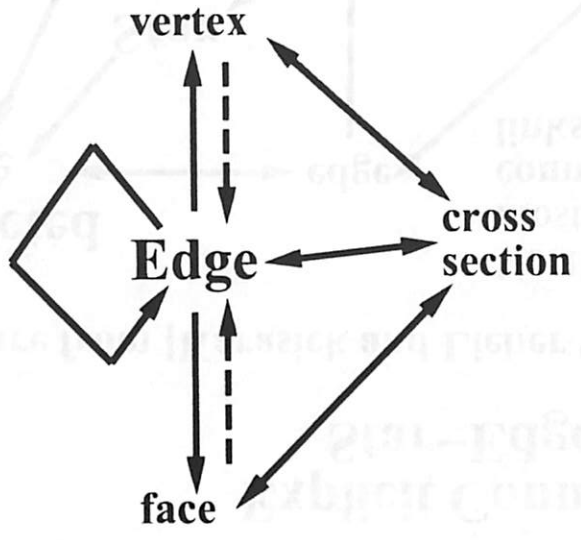
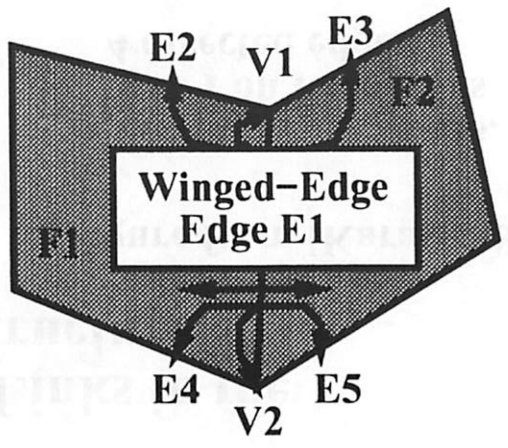


Figure from [Foley90]

Edge E1 has 8 pointers.
In addition, V1, V2, E2, E3, E4, E5, F1, F2 all have back pointers to E1.



RHW −UCB TCAD

Figure 4.6

Explicit Connectivity Links in the Star−Edge Data Structure

Figure from [Karasick and Lieber 90]

Vertices store most of the connectivity links via Stars.

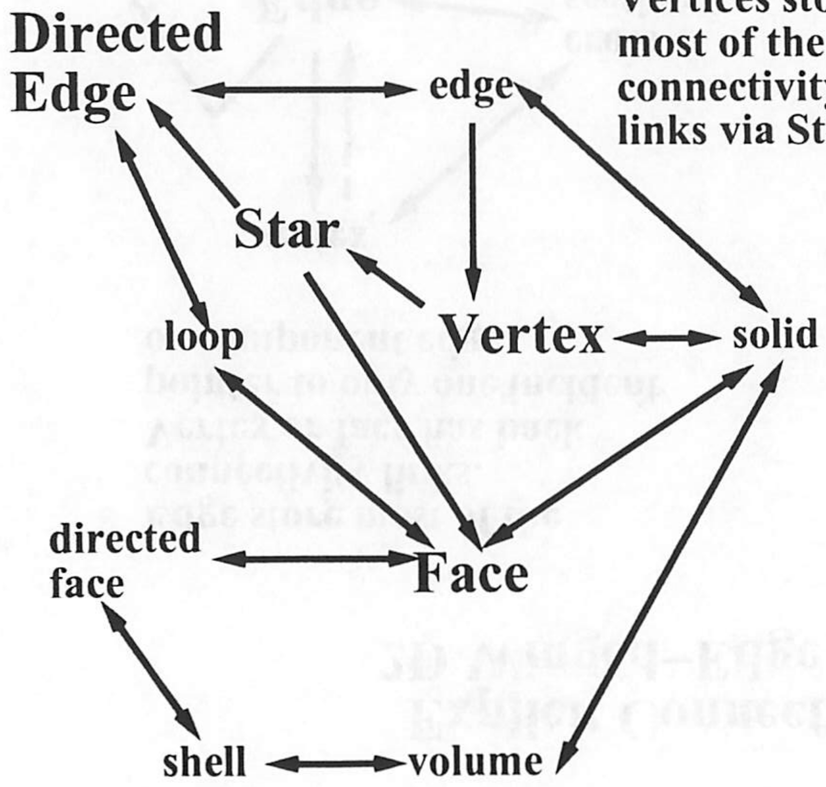
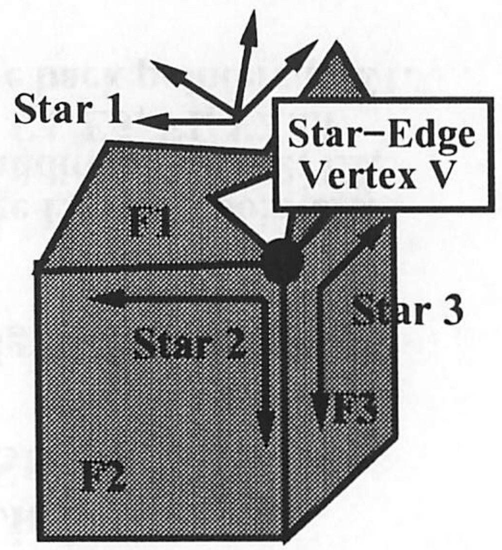


Figure from [Karasick88]

Vertex V has 3 stars. Star 1 on Face 1 has 4 directed edges.



RHW −UCB TCAD

4.4. FACE-FACE INTERSECTION SORTING

Face-Face Intersection Sorting groups faces by spatial locality, such that face-face intersections are only computed between faces within the same proximity. Without Face-Face Intersection Sorting, each surface facet is checked for intersections against every other surface facet. Consequently, $O(N^2)$ intersection checks are required to find all facet-facet intersections. On the other hand, in a typical topography simulation, each surface facet usually only intersects a few other surface facets. Therefore, without face-face intersection sorting, most of the $O(N^2)$ run time would be consumed by extraneous intersection calculations between far apart surface facets.

With **Face-Face Intersection Sorting**, each face-face intersection operation can be performed in $O(N^{\frac{3}{2}})$ to $O(N\log N)$ time. Implementing Face-Face Intersection Sorting involves adapting spatial data tructure parameters such that most spatial partitions contain only a few (less than 5) surface facets. Within each spatial partition, pair-wise face-face intersection computations require $O(n^2)$ time, where n is the number of the facets in each spatial partition. Since only a few facets are present in each partition, face-face intersections over all spatial partitions can be found in a running time dominated by the bigger of the two running times: 1) Preprocessing time used to sort faces into spatial partitions, and 2) Pair-wise intersection time used to visit each spatial partition and compute pair-wise intersections.

As examples, this section describes partitioning strategies for efficient 3D IC topography simulation using three spatial data structures currently implemented in TCAD applications: **Cell decomposition, Octrees** [19], and **BSP trees** [20]. Figure 4.7 [21] illustrates examples of 2D polygons partitioned using these data structures. The left of Figure 4.7 shows the **cell decomposition** data structure. This data structure uniformly divides the simulation space into rectilinear cells, and classifies each cell as **air** (empty), **material** (filled), or **surface** (partially filled). Using the SAMPLE-3D **cell decomposition** data structure, Scheckler had implemented a **cell deloop** operation that can run in $O(N^{\frac{3}{2}})$ time [3]. To achieve this run time, the cell segment length is set equal to the ideal segment length in the surface mesh. This setting enables most cells to contain a few facets. Since SAMPLE-3D cell contents are incrementally updated during surface advancement, face-face intersection operation time is dominated by the time used to perform pair-wise intersections in $O(N^{\frac{3}{2}})$ cells.

The middle of Figure 4.7 illustrates the **octree** data structure. This data structure recursively partitions the simulated surface into octants that are empty, filled, or containing a few facets. Using the SAMPLE-3D **octree** data structure, Helmsen had implemented an **octree deloop** operation that can run in *O(NlogN)* time [4]. In [4], it was shown that an 1:1 ratio between the octant (i.e. octree leaf) segment length and ideal facet segment length achieves the *O(NlogN)* running time while minimizing the number of octree nodes. Using the **octree** data structure, face-face intersection operation time is dominated by the preprocessing time used to sort surface facets into the octree. Since each tree insertion requires *O(logN)* time, *O(NlogN)* time is required to insert all *O(N)* facets into the octree.

The right of Figure 4.7 depicts the **BSP tree** data structure. This data structure subdivides space by recursively splitting and inserting facets as **in front of** or **behind** some initial (root) facet. Using the polygon in the right of Figure 4.7 as an example, BSP sorting begins by selecting facet **a** as the initial (root) facet, and grouping the remaining facets as two sets of facets: { **b, c+f, d+g, e, h, i** } and {**j, k**}. In this step, the facet that initially contains facets **d**, **g**, and **k** is first split into two facets **d+g** and **k**. The sorting process is then recursively applied to the two facet sets, with facet **b** as the initial (root) facet for the first set, and facet **j** as the initial (root) facet for the second set. Using this sorting strategy, each surface facet is split and inserted as a BSP tree **node**. The BSP tree **leaves** represent half-spaces that are either **air** (empty) or **material** (filled).

An *O(NlogN)* time **BSP tree deloop** operation can be implemented by inserting regularized surface facets into a BSP tree. Given a regularized mesh with N vertices and *O(N)* facets, the **BSP tree deloop** operation can insert all facets into a balanced binary search tree of height *O(logN)* in *O(NlogN)* time. During tree insertion, each intersecting facet can be split into facets that are on a valid or looping surface. An *O(N)* time traversal over all facets can be performed after tree insertion to remove looping surfaces.

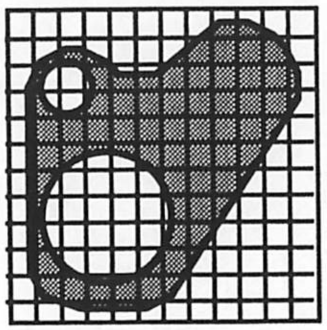
Partitioning Strategies for Spatial Data Structures

Figures from [Foley90]

Cell Decomposition

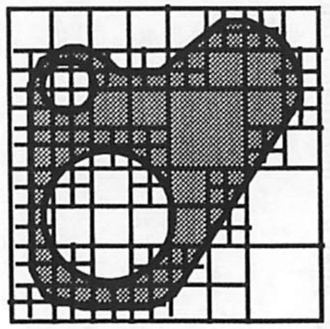
Simulation space is uniformly partitioned.

For sufficiently high resolution, each cell can contain zero or a few edges.



Quadtree / Octree

Simulation space is recursively partitioned into quadrants/octants until each quadrant/octant contains zero or a few edges/faces.



BSP Tree

In 2D, edges are inserted, split, and sorted in back–to–front ordering until all space (air or material) occupies a tree leaf. Each tree node contains an edge.

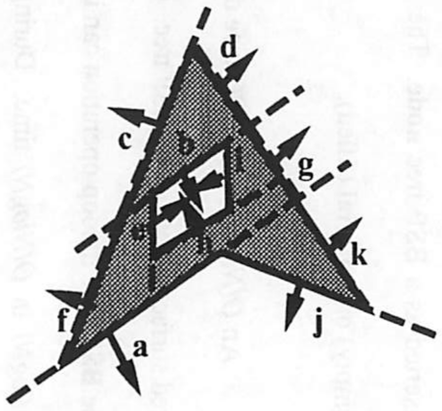


Figure 4.7

4.5. RAY-FACE INTERSECTION SORTING

Ray-Face Intersection Sorting groups faces by spatial locality, such that ray-face intersections are only computed between faces that lie in the ray's proximity. Without Ray-Face Intersection Sorting, each ray must be checked for intersection against every surface facet. Since a surface mesh with N vertices has *O(N)* facets, *O(N)* intersection checks are required to find all ray-face intersections. On the other hand, in a typical IC topography simulation, each ray usually only intersects a few surface facets. Therefore, without Ray-Face Intersection Sorting, most of the *O(N)* run time would be consumed by extraneous intersection calculations between the ray and far away facets.

With **Ray-Face Intersection Sorting**, each ray-face intersection operation can be performed in $O(N^{\frac{1}{2}})$ to *O(logN)* time. The implementation of Ray-Face Intersection Sorting involves adapting spatial data structure parameters such that most spatial partitions contain only a few (less than 5) surface facets. In addition, there is a need to amortize spatial data structure initialization cost over large numbers of ray tests.

For example, using the SAMPLE-3D **cell decomposition** data structure, Scheckler had implemented a **cell line-of-sight visibility test** that can run in $O(N^{\frac{1}{2}})$ time [3]. Again, to achieve this run time, the cell segment length is set equal to the ideal segment length in the surface mesh. Since SAMPLE-3D cell contents are incrementally updated during surface advancement, and the same SAMPLE-3D cells are used to perform all ray tests within each time step, ray-face intersection operation time is dominated by the time used to perform pair-wise intersections in $O(N^{\frac{1}{2}})$ cells.

In most general-purpose geometry servers, such as the IBM Geometry Engine, dynamically allocated spatial data structures are implemented to facilitate face-face intersection computations. To share spatial data structure initialization costs between ray-face intersection operations and face-face intersection operations, it may be advantageous for a geometry server to implement both operations using the same spatial data structure. This data structure sharing can be accomplished through two straightforward changes to the geometry server. These server changes reflect differences in input data and frequencies between these two operations.

First, spatial data structures need to be adapted to intersect and store ray segments. Many efficient algorithms exist to compute pair-wise ray-face intersections (e.g. see SAMPLE-3D's algorithm for computing ray-triangle intersections [3]). For **cell decompositions**, a test ray need to be divided into about $O(N^{\frac{1}{2}})$ ray segments contained by various cells. For **octrees** or **BSP trees**, a test ray can be efficiently represented using a pointer to the backmost facet or the frontmost facet hit by the ray.

Secondly, memory management of spatial data structures need to account for the different lifetimes of surface facets and rays. Within each simulation time step, surface facets are assumed to be immobile. Therefore, until the next surface advancement, the spatial sorting of surface facets generated during boundary deformation can be stored as a persistent surface facet database for repeated ray tests. On the other hand, ray position and direction changes with every test. Since millions of tests are used at each time step, test rays need to be stored as dynamically allocated ray segment lists.

4.6. LOCALIZED DEFORMATION

Localized Deformation provides facet pushing to efficiently update facet positions and connectivity links of moving solid boundaries in IC topography simulation. Without Localized Deformation, worst case $O(N^2 \log N)$ time might be used to update solid boundaries at each simulation time step. As discussed in Section 4.2, general-purpose geometry servers, such as the IBM Geometry Engine, ensures solid validity during boolean set operations by duplicating input solid geometry components and connectivity links. As a result, an IBM Geometry Engine boolean set operation requires $O(D_a D_b log(D_a D_b))$ time, where D_a and D_b are the number of face-edge incidences in input Solids A and B [18]. For the special case of constructing an aggregate deformation volume, after half (i.e. $\frac{O(N)}{2}$) of the merge operations have been performed, the number of face loops in the intermediate aggregate deformation volume (D_a or D_b) becomes $O(N)$. Consequently, for the remaining half (i.e. $\frac{O(N)}{2}$) of the merge operations, $O(N)$ geometry components and connectivity links are duplicated. This results in a worst-case run time of $O(N^2 \log N)$.

With **Localized Deformation**, an aggregate deformation volume can be constructed in about $O(N)$ to $O(N log N)$ time. Figure 4.8 illustrates two types of special-purpose boolean set operations that can be used to support Localized Deformation: **Surface-based boolean set operations**, and **Localized boolean set operations**. As depicted in the left and middle 1of Figure 4.8, **surface-based boolean set operations**, such as **deloop**, can play two important roles in simulating solid boundary deformation. First, **deloop** can be used in the traditional sense to remove extraneous loops after surface advancement. Secondly, as demonstrated by Sefler in [22], **deloop** can be used to merge together the initial surface (Surface 1), the delooped surfaces (Surfaces 2 and 3), and the simulation boundaries (Surfaces 4 and 5) into an aggregate deformation volume. Since each **deloop** operation can be performed in *O(NlogN)* time, the aggregate deformation volume can be constructed in *O(NlogN)* time.

Localized boolean set operations [23] can avoid extraneous connectivity links duplication by using the smaller input solid as a spatial filter for delimiting perturbed faces in the larger input solid. As depicted in the right of Figure 4.8, **localized boolean set operation** can create new facets, and resolve facet-facet intersections within a spatial filter defined by the facet sweep. This effectively prevents unperturbed faces in the intermediate aggregate deformation volume from participating in the operation. Mantyla showed that **localized boolean set operations** run in *O(n log n)* time, where n is the number of faces in the smaller input solid [23]. For boundary deformation, the smaller argument is either a tetrahedron (4 faces) or a triangular prism (5 faces). Therefore, boundary deformation using repeated **localized set operations** can run in a more *O(N)*-like *O(N * n log n)* time.

Figure 4.8

Boolean Set Operations for Localized Deformation

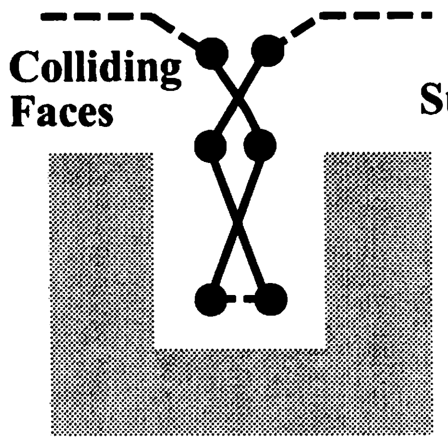
Surface–Based Boolean Set Operations (Deloop)

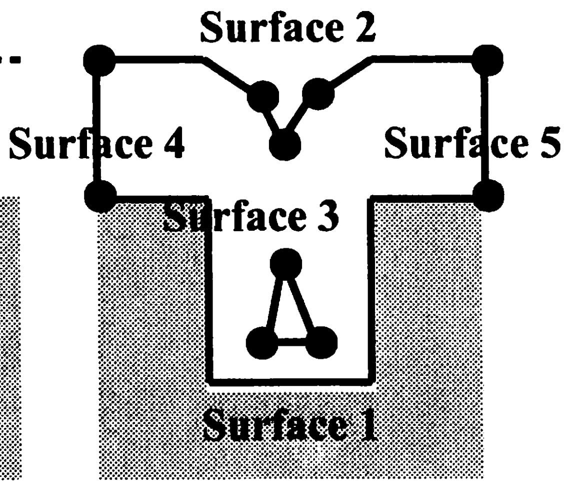
Remove extraneous loops from surface advancement.

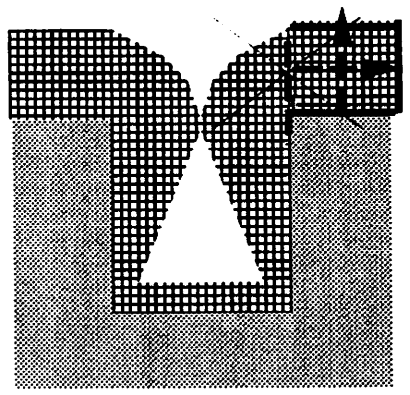
Extract solid boundaries.

Localized Boolean Set Operations

Create new facets, push them, and resolve inter–sections within delimiters.







RHW – UCB TCAD

4.7. CONCLUSIONS

This chapter showed that the performance of geometrical operations in 3D IC topography simulation are linked to four **essential geometry server constructs**. These constructs can be implemented using conventional connectivity and spatial data structures, and special-purpose boolean set operations. The implementation of these constructs constitutes a necessary but not sufficient condition for efficient 3D IC topography simulation. This fact was demonstrated by comparing the theoretical performance of geometrical operations implemented with and without the constructs. Since the constructs were shown to be necessary for efficient 3D simulation, they may be used to screen potential servers.

Section 4.3 described how **Explicit Connectivity** improves surface connectivity query performance from *O(N)* time to *O(1) (constant)* time per query. For Explicit Connectivity, the key implementation principle is to choose a low topological order geometry component, such as an edge or a vertex, to store most of the connectivity links. In particular, connectivity links are established from special geometry components to all of their adjacent components.

As an example, Section 4.3 introduced the 2D **winged-edge** data structure, which uses **edges** to store connectivity links to component vertices, ajdacent edges, and incident faces. To facilitate connectivity queries, each vertex or face contains a back pointer to an adjacent edge. As another example, Section 4.3 described the 3D **star-edge** data structure, which was implemented in the IBM Geometry Engine. The **star-edge** data structure uses **vertices** to store connectivity links to incident edges and incident faces. At each vertex, incident edges are organized into **stars**. Each **star** contains links to an incident face, and the incident edges on that face.

Section 4.4 described how **Face-Face Intersection Sorting** can improve the performance of face-face intersection operations from *$O(N^2)$* time to *$O(N^{\frac{3}{2}})$* time for grid-based data structures, and *$O(N\log N)$* time for tree-based data structures. In using 3D spatial data structures to implement Face-Face Intersection Sorting, the key implementation principle is to set the cell size or the tree depth, such that each spatial partition contains a few surface facets. A practical rule-of-thumb is to set the cell/octant segment length equal to the ideal facet segment legnth. As discussed in Section 4.4, using this technique, SAMPLE-3D's **cell deloop** can be performed in *$O(N^{\frac{3}{2}})$* time, and SAMPLE-3D's **octree deloop** can be performed in *$O(N\log N)$* time. Section 4.4 also briefly described the algorithm for an *$O(N\log N)$* time **BSP tree deloop** operation.

Section 4.5 described how **Ray-Face Intersection Sorting** can improve the performance of **point location tests** or **line-of-sight visibility tests** from *$O(N)$* time to *$O(N^{\frac{1}{2}})$* time for grid-based data structures, and *$O(\log N)$* time for tree-based data structures. Similar to Face-Face Intersection Sorting, in using 3D spatial data structures to implement Ray-Face Intersection Sorting, the key implementation principle is to set the cell size or tree depth such that each spatial partition contains a few surface facets. In addition, there is the need to amortize spatial data structure initialization costs over large number of ray tests. As discussed in Section 4.5, using these techniques, SAMPLE-3D's **cell line-of-sight visibility test** can find all ray-face intersections in *$O(N^{\frac{1}{2}})$* time.

Section 4.5 also described a two-step server modification for adapting **Face-Face Intersection Sorting** data structures to simultaneously support **Ray-Face Intersection Sorting**. The goal of this server modification is to share spatial data structure initialization cost between face-face and ray-face intersection operations. The two-step modification involves: 1) Creating geometric utilities to subdivide rays into ray segments, and intersect ray segments with faces; 2) Making the **Face-Face Intersection Sorting** data structure a part of the persistent geometry server data. Most geometry servers dynamically allocate 3D spatial data structures during boolean set operations. Therefore, Step 2) may represent a significant change in geometry server memory management.

Section 4.6 described how **Localized Deformation** can improve the performance of boundary deformation from $O(N^2 logN)$ time to $O(NlogN)$ time or $O(N * n \log n)$ time, where n is the number of faces in a facet swept deformation volume. Section 4.6 described two types of special-purpose **boolean set operations** that exploits spatial and temporal locality in IC topography simulation to avoid extraneous duplications of geometry components and connectivity links. The first approach involved pushing surface facets, and using **surface-based boolean set operations**, such as **deloop** operations, to construct a valid aggregate deformation volume. As an example, Section 4.6 discussed how SAMPLE-3D's **solid extraction** operation uses **deloop** to generate solids in $O(NlogN)$ time.

The second approach involved using **localized boolean set operations. Localized boolean set operations** use triangular deformation prisms as spatial filters. Within the spatial filter, the operation can push a subset of solid model faces, and update connectivity links between these faces. Section 4.6 discussed Mantyla's implementation. Mantyla showed that the performance of a **localized boolean set operation** can be made to depend only on n. As a result, boundary deformation using repeated **localized boolean set operations** can be performed in a more $O(N)$-like $O(N * n \log n)$ time.

REFERENCES FOR CHAPTER 4

[1] M. Karasick, D. Lieber, "Schemata for Interrogating Solid Boundaries," ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[2] K.K.H. Toh, A.R. Neureuther, E.W. Scheckler, Algorithms for simulation of three-dimensional etching. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, May 1994, vol.13, (no.5):616-24.

[3] E.W. Scheckler, A.R. Neureuther, Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1994, vol.13, (no.2):219-30.

[4] J.J. Helmsen, A comparison of three-dimensional photolithography simulators. Ph.D. Thesis, UC Berkeley, UCB/ERL M95/25, 1995.

[5] P. Conti, N. Hitschfeld, W. Fichtner, Omega -an octree-based mixed element grid allocator for the simulation of complex 3-D device structures. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Oct. 1991, vol.10, (no.10):1231-41.

[6] D. Yang, Mesh generation and information model for device simulation. Ph.D. Thesis, Stanford University, June 1994.

[7] P. Lloyd, C.C. McAndrew, M.J. McLennan, S.R. Nassif, and others. Technology CAD at AT&T. Microelectronics Journal, March 1995, vol.26, (no.2-3):79-97.

[8] B.F. Naylor, Interactive solid geometry via partitioning trees. Proceedings. Graphics Interface '92, Vancouver, BC, Canada, 11-15 May 1992. p. 11-18.

[9] F. Jones, J. Paraszczak, RD3D (computer simulation of resist development in three dimensions). IEEE Transactions on Electron Devices, Dec. 1981, vol.ED-28, (no.12):1544-52.

[10] C.H. Sequin, Computer simulation of anisotropic crystal etching. Sensors and Actuators A (Physical), Sept. 1992, vol.A34, (no.3):225-41..

[11] Hung Liao, T.S. Cale, Three-dimensional simulation of an isolation trench refill process. Thin Solid Films, 15 Dec. 1993, vol.236, (no.1-2):352-8.

[12] S. Tazawa, F.A. Leon, G.D. Anderson, T. Abe, and others. 3-D topography simulation of via holes using generalized solid modeling. Proceedings of IEEE International Electron Devices Meeting, San Francisco, 13-16 Dec. 1992. p. 173-6.

[13] E. Strasser, S. Selberherr, Algorithms and models for cellular based topography simulation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1995, vol.14, (no.9):1104-14.

[14] D. Adalsteinsson, J.A. Sethian, A level set approach to a unified model for etching, deposition, and lithography. II. Three-dimensional simulations. To appear Journal of Computational Physics, 1995.

[15] J.A. Sethian, Theory, algorithms, and applications of level set methods of propagating interfaces. To appear Acta Numerica, 1995.

[16] H. Edelsbrunner, *Algorithms in combinatorial geometry*, Springer-Verlag, Berlin, 1987.

[17] B.G. Baumgart, Geometric modeling for computer vision. Ph.D. Thesis, Stanford University, 1974.

[18] M.S. Karasick, On the representation and maniuplation of rigid solids. Ph.D. Thesis, McGill University (also Cornell University TR 89-976), 1989.

[19] H. Samet, *Applications of spatial data structures*. Addison-Wesley Publishing Co., 1990. (Note: Survey of octree algorithms and applications.)

[20] B. Naylor, Binary space partitioning trees as an alternative representation of polytopes. Computer Aided Design, May 1990, vol.22, (no.4):250-2.

[21] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Ed., Addison-Wesley Publishing Co., 1987.

[22] J. Sefler, 3D Surface Modeling Utilities for use in TCAD, MS Thesis, UC Berkeley, October 28, 1995.

[23] M. Mantyla, *Introduction to solid modeling*. Rockville, MD, USA: Computer Science Press, 1988.

CHAPTER 5

IBM GEOMETRY ENGINE CONSTRUCTS

5.1. INTRODUCTION

The IBM Geometry Engine [1] as a modern solid modeler is a prime candidate to be used to support IC topography simulation. However, even in this modeler not all of the geometry server constructs essential to IC topography simulation are implemented. This chapter describes IBM Geometry Engine constructs, and discusses how they might degrade 3D simulation performance. The IBM Geometry Engine provides Explicit Connectivity and Face-Face Intersection Sorting, but lacks Ray-Face Intersection Sorting and Localized Deformation. Table 5.1 summarizes IBM Geometry Engine constructs. In Table 5.1, the constructs supported by the IBM Geometry Engine are listed in the leftmost column, and the corresponding data structures are listed in the middle column. The rightmost column of Table 5.1 lists the connectivity query performance , and the face-face intersection operation operation that can be achieved through the use of IBM Geometry Engine constructs.

Developed at IBM Research during the early 1990's, the IBM Geometry Engine was primarily designed as an Explicit Connectivity geometry server that could replace the GDP solid modeler [3][4]. When GDP was used in the early 1980's to simulate 3D device structures in the OYSTER [5] system, its inability to explicitly represent material interfaces became a major barrier towards rigorous IC topography simulation. As shown in the first row of Table 5.1, the IBM Geometry Engine improves over the GDP solid modeler by providing Explicit Connectivity through two data structures: The **star-edge** data structure and the **star-edge schema** [1]. As previously described in Chapter 4, the **star-edge** data structure is a 3D extension of the 2D **winged-edge** data structure. With respect to IBM Geometry Engine data organization, the **star-edge** data structure stores geometry components and connectivity links internal to the server, and cannot be directly accessed by application programs.

Section 5.2 describes the **star-edge schema** for accessing IBM Geometry Engine geometry components and connectivity links. Based on connectivity services requested by the application program, the **star-edge schema** selectively extracts server geometry components and connectivity links. The extracted components and links are then tranformed by the **star-edge schema** into a format that hides the connectivity links in the **star-edge** data structure. Since server data may be updated at each simulation time step, the frequent transfer of geometry components and connectivity links from the **star-edge** data structure to the **star-edge schema** may degrade 3D simulation performance. In Section 5.2, the efficiency of this data transfer will be studied with respect to the connectivity service requirements of 3D IC topography simulation.

Section 5.3 describes the **bucket sorting of face bounding boxes** data structure for supporting Face-Face Intersection Sorting in the IBM Geometry Engine. Conventionally, 3D Face-Face Intersection Sorting has been implemented using spatial data structures that sort faces into 3D spatial partitions, such as **octrees** or **BSP trees.** The IBM Geometry Engine uses a spatial data structure that sorts face bounding boxes into buckets along the principal (*x*-, *y*-, and *z*-) axes, or 1D spatial partitions. As shown in the second row of Table 5.1, in general, the IBM Geometry Engine can use this data structure to find all face-face intersections in *empirical O(N)* time [2]. In Section 5.3, the efficiency of the **bucket sorting of face bounding boxes** data structure will be studied for the special case of deformation volume stitch-back after an etching or deposition process step.

IBM Geometry Engine Constructs

Construct	Data Structures and Algorithms		Reported Performance with Construct
Explicit Connectivity	Star–Edge Data Structure (Section 4.3)	Star–Edge Schema (Section 5.2)	O(1) time per query.
Face–Face Intersection Sorting	(1D) Bucket Sorting of Face Bounding Boxes (Section 5.3)		empirically O(N) time per solid intersection.

Table 5.1

5.2. EXPLICIT CONNECTIVITY: THE STAR-EDGE SCHEMA

The **star-edge schema** provides a convenient interface for accessing IBM Geometry Engine geometry components and connectivity links by hiding the connectivity links in the underlying **star-edge** data structure. Figure 5.1 illustrates the geometry components and connectivity links in the **star-edge schema**. In Figure 5.1, italicized objects are **iterators**, or objects that iterates through the elements of a linked list, and returns a pointer to the traversed element during each iteration. As shown in Figure 5.1, connectivity links in the underlying **star-edge** data structure are transformed into a generic set of iterators: *VolumesOfSolid(Vertex|Edge|Face)*, *FacesOfSolid(Vertex|Edge)*, and *EdgesOfSolid(Vertex)*. For example, to obtain the edges incident on schema vertex V, the *EdgesOfSolid(V)* object is activated. This generic interface completely hides the internal data organization of the **star-edge** data structure.

Based on connectivity services requested by the application program, the **star-edge schema** can selectively extract a topological subset of server geometry components and connectivity links. In some cases, the **star-edge schema** might split its copy of the underlying **star-edge** edges and faces to reduce the topological complexity of the schema solid model. Figure 5.2 illustrates the extraction of a simplified **star-edge schema** 2D solid model, from a topologically complex **star-edge** structure 3D solid model. As shown in Figure 5.2, the schema splits the hour-glass polygonal face F in the 3D solid model, into two triangular faces F1 and F2 in the 2D subset.

Due to topological subset selection and simplification, connectivity links in the **star-edge** data structure cannot be directly transferred to the **star-edge schema**. As an example, Figure 5.2 shows how these features prevents the direct transfer of connectivity links from

star-edge vertex V to **star-edge schema** vertex V. In the 3D solid model, the **star-edge** vertex V has an incident edge that goes along the *+y* direction, and an incident face F that is in the shape of an hour-glass. Due to topological subset selection, the *+y* direction edge has no corresponding schema edge. Since the schema had split its copy of face F, face F has no directly corresponding schema face.

Instead of directly transferring **star-edge** connectivity links, the **star-edge schema** establishes connectivity links between schema geometry components through a two-step procedure. First, for each selected **star-edge** vertex, all of its incident **star-edge** edges and faces are found by traversing vertex stars. Then, for each **star-edge** component incident on the selected vertex, a linear search is carried out on the appropriate **star-edge schema** component list to identify the **star-edge schema** equivalent(s). In the special case of 3D IC topography simulation, all **star-edge** vertices, edges, and faces need to be represented in the **star-edge schema**. Therefore, for 3D IC topography simulation, the initialization of schema connectivity links may require { *N* schema vertices } * { 3 to 6 incident edges and faces } * { $O(N)$ time per linear search on schema component list of size $O(N)$ } = $O(N^2)$ time.

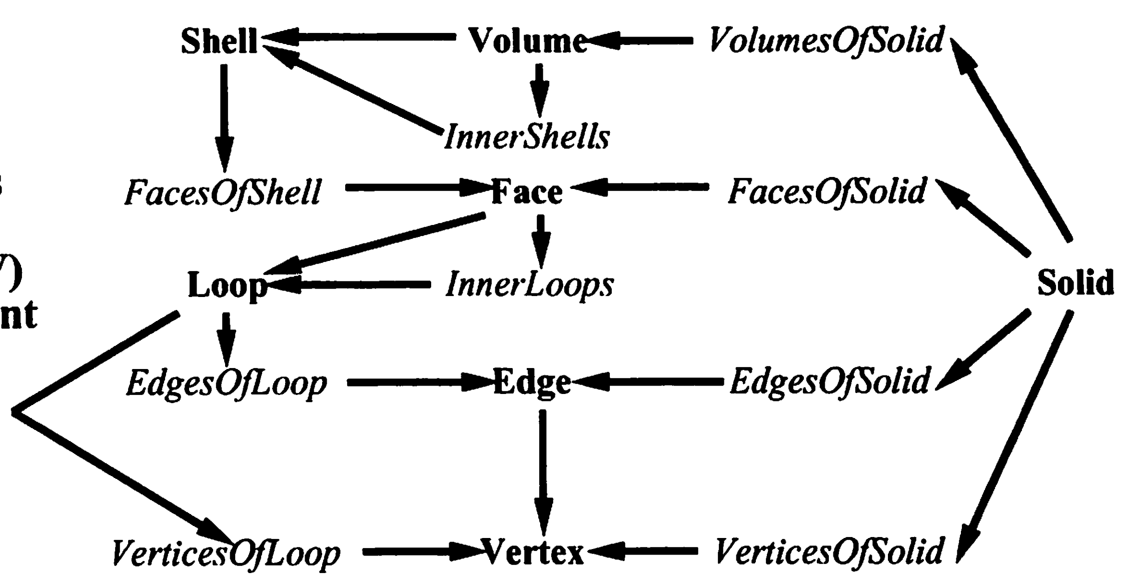
Figure 5.1

Explicit Connectivity Links in the Star–Edge Schema

Geometry component iterators hide connectivity links in the underlying Star–Edge DS.

Examples:
- EdgesOfSolid() returns all edges in the solid.
- EdgesOfSolid(Vertex V) returns all edges incident on V.

[Figure from Karasick and Lieber 90]



Connectivity Links in Star–Edge Vertex V cannot be directly transferred to Schema Vertex V.

[Figure adapted from Karasick and Lieber 90]

Due to Topological Subset Selection, there is no corres–poding schema edge going into the page.

Due to Face Splitting, there are two schema faces corresponding to Star–Edge Face F.

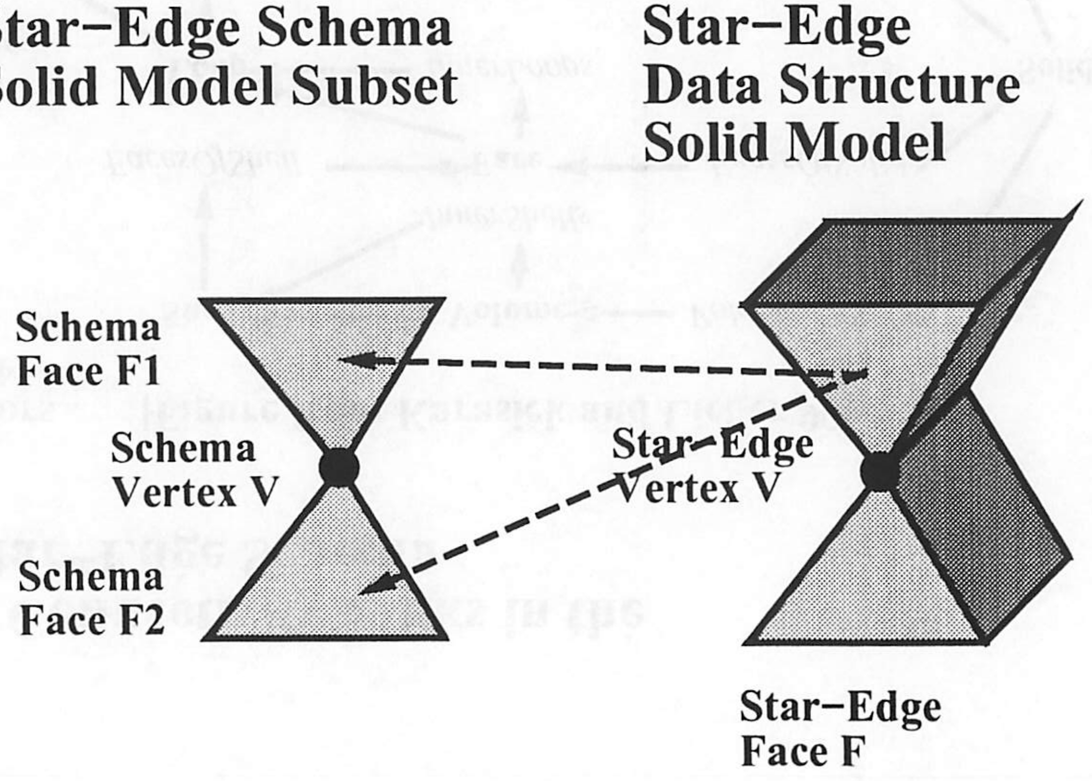


Figure 5.2

5.3. FACE-FACE INTERSECTION SORTING: BUCKET SORTING OF FACE BOUNDING BOXES

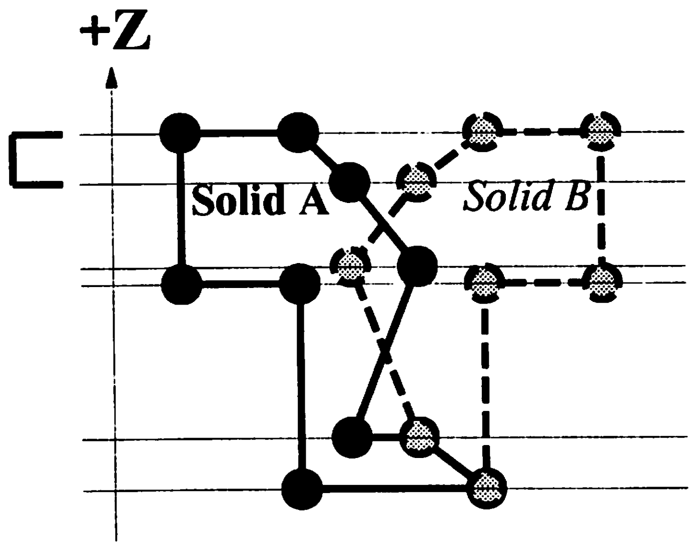
Through the use of the **bucket sorting of face bounding boxes** data structure, the IBM Geometry Engine supports an efficient expected *O(N)* time face-face intersection operation. In the left of Figure 5.3, two overlapping solids, Solid *A* and Solid *B*, are being merged by boolean set operations. The bucket sorting data structure sorts solid faces into 1D buckets along the principal *z*-axis. As shown in the right of Figure 5.3, each bucket contains faces whose bounding boxes are partially or completely extended into the bucket. In general, to achieve expected *O(N)* time performance, buckets need to be created along a random direction, and projected onto the principal axes. Alternatively, the IBM Geometry Engine creates buckets along the principal axis with the greatest extent.

The right of Figure 5.3 illustrates how the bucket sorting data structure effectively helps the IBM Geometry Engine avoid extraneous face-face intersections. First, the IBM Geometry Engine checks for overlaps between face bounding boxes. Pairwise face intersections are then computed only between faces with overlapping bounding boxes. For the bucket shown in the right of Figure 5.3, 9 face bounding box intersection checks would be performed. Since these checks would not find any bounding box overlap, there would be no need to compute pairwise face intersections in this bucket.

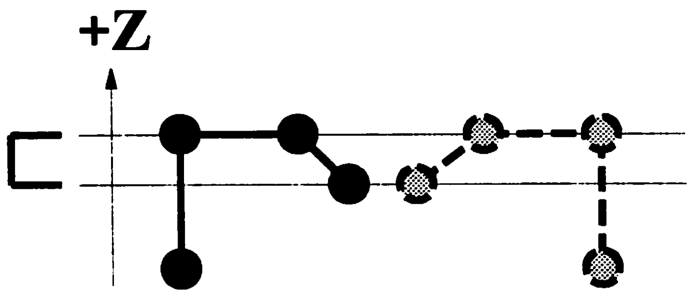
For the special case of deformation volume stitch back in IC topography simulation, the bucket sorting data structure might result in $O(N^2)$ face bounding box intersection checks. When an aggregate deformation volume is stitched onto a wafer geometry after a deposition process, the wafer top surface is intersected against itself. Due to the predominantly planar nature of IC topographies, many of the O(N) surface facets may be co-planar, and parallel to the xy-plane. Therefore, the *z*-axis bucket located near the wafer top surface may contain up to *O(N)* co-planar faces. As a result, for this bucket, the IBM Geometry Engine may invoke $O(N^2)$ face bounding box intersection checks.

"Z" Bucket Sorting of Face Bounding Boxes for Computing Face–Face Intersections

Divisions demarcating "Z" Buckets



3 x 3 = 9 face bounding box intersection checks.
–> No bbox overlap.



No bbox overlap.
–> No need to pairwise intersect faces.

Figure 5.3

5.4. CONCLUSIONS

This chapter described the **essential geometry server constructs** implemented in the IBM Geometry Engine. As discussed in Section 5.1, the IBM Geometry Engine supports two out of the four essential constructs. The IBM Geometry Engine implemented two Explicit connectivity data structures. The **star-edge** data structure stores connectivity information inside the IBM Geometry Engine. The **star-edge schema** provides read-only access to server connectivity information. The IBM Geometry Engine also implemented an unconventional **Face-Face Intersection Sorting** data structure that sorts face bounding boxes into 1D spatial partitions. The IBM Geometry Engine does not support **Ray-Face Intersection Sorting** and **Localized Deformation.**

At each time step of a 3D IC topography simulation, Section 5.2 showed that the IBM Geometry Engine requires $O(N^2)$ time to transfer connectivity links from the **star-edge** data structure to the **star-edge schema**. Through the **star-edge schema**, application programs can request a subset of the server components, and simplify the topology of the subset components, such as subdividing an hour-glass face into two triangular faces. Since the **star-edge schema** may contain a modified subset of the server components, schema components that corresond to server connectivity links have to be found using linear searches through schema components lists. For 3D IC topography simulation, a topography with N vertices has $O(N)$ edges and faces. Therefore, the **star-edge schema** contains several lists of $O(N)$ vertices, edges, and faces. For each schema vertex, the IBM Geometry Engine needs $O(N)$ time to establish its connectivity links. Overall, connectivity links are transferred in $O(N^2)$ time.

For typical IC topographies with about 10,000 facets, the efficiency gained in performing schema connectivity queries might be swamped over by the $O(N^2)$ time needed to establish schema connectivity links. As discussed in Chapter 4, a typical IC topography simulation time step may involve up to *O(N)* connectivity queries. Since each **star-edge schema** connectivity query requires only *O(1)* (constant) time, all connectivity queries can be efficiently performed in *O(N)* time. In Chapter 6, as a part of IBM Geometry Engine **standardized performance tests**, the run times of **star-edge schema** connectivity links initialization and connectivity queries will be compared on planar topographies.

For the special case of deformation volume stitch back in IC topography simulation, the **bucket sorting of face bounding boxes** data structure might result in $O(N^2)$ face bounding box intersection checks. When an aggregate deformation volume is stitched onto a wafer geometry after a deposition process, the wafer top surface is intersected against itself. Due to the predominantly planar nature of IC topographies, many of the O(N) surface facets may be co-planar, and parallel to the xy-plane. Therefore, the *z*-axis bucket located near the wafer top surface may contain up to *O(N)* co-planar faces. As a result, for this bucket, the IBM Geometry Engine may invoke $O(N^2)$ face bounding box intersection checks. In Chapter 6, as a part of IBM Geometry Engine standardized performance tests, the run time performance impact of this topography simulation induced inefficiency will be characterized using vertical deposition on planar and non-planar 3D topographies.

REFERENCES FOR CHAPTER 5

[1] M. Karasick, D. Lieber, "Schemata for Interrogating Solid Boundaries," ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[2] M.S. Karasick, On the representation and maniuplation of rigid solids. Ph.D. Thesis, McGill University (also Cornell University TR 89-976), 1989.

[3] M.A. Wesley, L.I. Lieberman, M.A. Lavin, D.D. Grossman, and others. A geometric modeling system for automated mechanical assembly. IBM Journal of Research and Development, Jan. 1980, vol.24, (no.1):64-74.

[4] R.N. Wolfe, M.A. Wesley, J.C. Kyle, Jr., F. Gracer, and others. Solid modeling for production design. IBM Journal of Research and Development, May 1987, vol.31, (no.3):277-95.

[5] G.M. Koppelman, M.A. Wesley, OYSTER: a study of integrated circuits as three-dimensional structures. IBM Journal of Research and Development, March 1983, vol.27, (no.2):149-63.

CHAPTER 6
PERFORMANCE EVALUATION OF THE IBM GEOMETRY ENGINE

6.1. INTRODUCTION

This chapter introduces **standardized performance tests** as a system tool for evaluating and guiding the use of centralized geometry servers in 3D IC topography simulation. Standardized performance tests are representative test cases of geometrical operations used in IC topography simulation. These tests are based on the need for essential geometry server constructs. The tests attempt to measure the effectivenss of implemented constructs, and characterize the performance impact of missing constructs. Performance test results, such as those in this chapter, can give insights to the design of auxiliary data structures which can compensate for ineffective or missing constructs.

Standardized performance tests provide a means for automatically tracking the progress in general-purpose servers for 3D IC topography simulation. Section 6.2 describes the BTU systems tools that were used to conduct IBM Geometry Engine [1] standardized performance tests. As will be shown in this chapter, considerable coding is necessary to interface the Geometry Engine for IC topography simulation. The IBM Geometry Engine standardized performance tests are designed to evaluate the effectiveness of IBM Geometry Engine constructs, and characterize the performance impact of not supporting Ray-Face Intersection Sorting and Localized Deformation. A preliminary version of these tests have been published in [2]. All test results reported in this chapter were obtained on an IBM RS/6000 Model 530 workstation with 32 MB RAM.

Section 6.3 describes the **Explicit Connecitivity Test**. To evaluate the efficiency of IBM Geometry Engine's **star-edge schema**, this test uses breadth first traversal of surface vertices to supply representative surface connectivity queries. Breadth first traversal involves recursively using the connectivity between a traversed vertex and its adjacent edges and faces to find connected vertex neighbors for continuing the traversal. Therefore, this traversal effectively exercises useful schema connectivity links. This test measures the CPU times used to perform breadth first traversal of all surface vertices on 3D planar topographies with 70 faces to 500 faces. As recommended in Chapter 5, the **Explicit Connectivity Test** also assesses the performance impact of the $O(N^2)$ time required to initialize schema connectivity links. In Section 6.3, the CPU times of schema connectivity queries are compared with that of schema connectivity links initialization.

Section 6.4 describes the **Face-Face Intersection Sorting Test**. To demonstrate the efficiency gained through IBM Geometry Engine's **bucket sorting of face bounding boxes** data structure, boolean set operations are used to glue 3D IC topographies with their vertical deposition volumes. In this test, the CPU times used to compute solid intersection curves are measured for 3D topographics with 100 surface triangles to 1,000 surface triangles. As recommended in Chapter 5, the **Face-Face Intersection Sorting Test** also characterizes the performance impact of using $O(N^2)$ face bounding box intersection computations to compute face-face intersections between planar topographies and their vertical deposition volumes. In Section 6.4, this performance impact is assessed by comparing the CPU times used to compute solid intersection curves on planar versus non-planar topographies.

Section 6.5 describes the **Ray-Face Intersection Sorting Test**. The first part of the **Ray-Face Intersection Sorting Test** evaluates the performance impact of not supporting Ray-Face Intersection Sorting in the IBM Geometry Engine. In this part of the test, material interface collision detection is emulated by applying **point location tests** to points on a vertical etch front. In the IBM Geometry Engine, the **point location test** is implemented by shooting out a ray from a query point towards the *-x*-direction, and using ray-face intersections to identify the vertex, edge, face, or volume that contains the query point. To characterize the performance impact of checking for intersections between a ray and *O(N)* faces, average CPU times are measured for IBM Geometry Engine **point location tests** on 3D planar and non-planar topographies with 300 to 3,000 faces.

The second part of the **Ray-Face Intersection Sorting Test** demonstrates the performance improvement that could be gained by using the **bucket sorting of face bounding boxes** data structure to support Ray-Face Intersection Sorting. Another purpose of this part of the test is to introduce **geometrical operation transformation** as a data organization method to compensate for the lack of Ray-Face Intersection Sorting. In this part of the test, **boolean set operations** between sliver tetrahedra and 3D IC topographies, replace **point location tests** on etch front points. Each sliver tetrahedron consists of an apex located near the center of the topography, and a vertical triangular base located to the left of the topography. For comparison with CPU times on IBM Geometry Engine **point location tests** (Part 1), CPU times are measured for solid intersection curve computations between a sliver tetrahedron, and 3D planar and non-planar topographies with 300 to 3,000 faces.

Section 6.6 describes the **Localized Deformation Test**. The first part of the **Localized Deformation Test** demonstrates the performance impact of not supporting Localized Deformation in the IBM Geometry Engine. In this part of the test, boundary deformation using triangular prisms is emulated by staircase construction from cubes. In the context of boundary deformation, a cube is computationally equivalent to a triangular prism generated by a triangular facet sweep. Just as merging each triangular prism introduces localized changes to the intermediate aggregate deformation volume, merging each cube introduces localized changes to the intermediate staircase. To assess the performance impact of repeatedly copying over unaffected facets and connectivity links in intermediate staircases, CPU times are measured for merging 100, 400, 900, and 1600 1 um x 1 um x 1 um cubes into staircases.

The second part of the **Localized Deformation Test** demonstrates how boolean set operations can be efficiently used to simulate boundary deformation by increasing input data granularity. Another purpose of this part of the test is to introduce **large grain surface decomposition** as a data organization method to compensate for the lack of Localized Deformation. In this part of the test, boundary deformation using large grain deformation volumes is emulated by staircase construction from $\sqrt{N}$-cube strips. Since each cube strip has $\sqrt{N}$ cubes, the number of **merge** operations is reduced by $\sqrt{N}$x. This in turn reduces by $\sqrt{N}$x the number of duplications of unperturbed facets and connectivity links. To demonstrate the performance improvement that could be gained by increasing input data granularity, CPU times are measured for merging 10 10-cube strips, 20 20-cube strips, 30 30-cube strips, and 40 40-cube strips into staircases.

6.2. STANDARDIZED PERFORMANCE TESTING IN THE BTU SYSTEM

This section introduces the BTU system utilities for constructing and manipulating these structures. In implementing standardized performance tests, it is shown that the demand of IC topography simulation requires considerable augmentation of the standard interfaces to general-purpose geometry servers. About 10,000 lines of C++ code are likely necessary to perform relatively basic geometrical operations in 3D IC topography simulation with any solid modeling package. For example, about 12,000 lines of C++ code are required to interface the IBM Geometry Engine for performance testing. Out of these 12,000 lines, a significant part performs mundane geometry construction and data mapping tasks, such as constructing tiled initial topographies, extracting surface faces, triangulating polygonal faces, and constructing vertical deformation volumes.

Three out of the four IBM Geometry Engine standardized performance tests use the planar stack and two-holes initial structures. Figure 6.1 plots a 2D cross section of the two-holes structure at y = 0. This figure depicts the dimensions and positions of the planar stack and two-holes structures to be used in the **Explicit Connectivity Test**, the **Face-Face Intersection Sorting Test**, and the **Ray-Face Intersection Sorting Test**. As shown in Figure 6.1, each initial structure consists of two layers: A 2 um thick silicon substrate, and a 0.6 um thick oxide layer. Each layer has a length of 6 um along the *x* direction, and a length of 4 um along the *y* direction (not shown). The silicon substrate is centered at (0,0,1), and the oxide layer is centered at (0,0,2.3). (The exception is in the **Face-Face Intersection Sorting Test**, which uses a 6 um thick substrate.) The holes in the two-holes structure are cut using two inverted cone stubs. Each cone stub has a top radius of 1 um, a bottom radius of 0.6 um, and a height of 0.6 um. The cone stubs are centered at (-1.5, 0, 2.3), and (1.5, 0, 2.3).

The BTU system provides several utilities for constructing the planar stack and two-holes initial structures, and vertically depositing various material layers on these structures. First, there is the **initial structure construction** utility. This utility creates planar stack and two-holes solid structures using the following boundary meshing parameters: 1) **NX** = the number of divisions per micron along the *x* direction of a layer, 2) **NY** = the number of divisions per micron along the *y* direction of a layer, 3) **NTheta** = the number of angular divisions in a cone stub, 4) **NSlices** = the number of vertical divisions in a cone stub. and 5) **NHoles** = the number of holes in the oxide layer. The upper left corner of Figure 6.2 illustrates a two-holes initial structure created by the **initial structure construction** utility using NX = 2, NY = 2, NTheta = 16, NSlices = 4, and NHoles = 2.

The **surface mesh extraction** utility extracts the surface faces of an IBM Geometry Engine solid model, and creates a surface mesh by triangulating each surface face. As an example, the lower left corner of Figure 6.2 depicts the triangular surface mesh extracted from the two-holes initial structure by the **surface mesh extraction** utility. To ease the sub-task of face triangulation, the IBM Geometry Engine's **face simplification** facility was disabled. Face simplification removes shared edges between adjacent co-planar faces, and may create polygons that have holes (i.e. non-simply-connected polygons). Disabling the **face simplification** facility can help maintain a finely meshed solid model that contains only simply-connected polygonal faces. As a result, surface faces can be easily triangulated by connecting face vertices.

The **vertical deformation volume construction** utility creates a vertical deformation volume by extruding an initial surface mesh along the *z* direction. In the BTU system, the mesh extrusion is implmented by copying the initial surface mesh to a moving surface mesh, uniformly increasing or decreasing the *z* components of the moving mesh vertices, and stitching the initial mesh and the moving mesh along their boundaries. As an example, the upper right corner of Figure 6.2 shows the 0.7 um polysilicon layer created from the two-holes initial surface mesh by the **vertical deformation volume construction** utility.

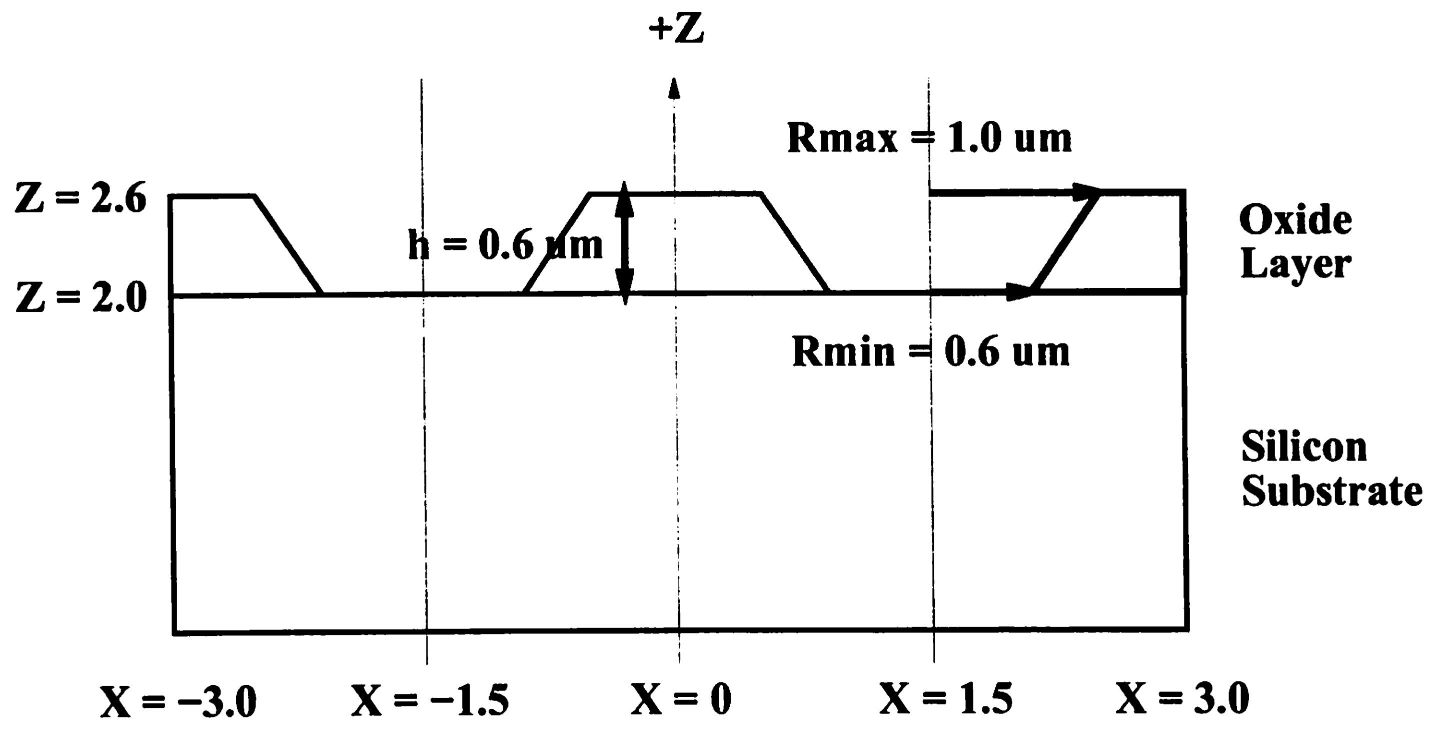
The **vertical deformation volume stitch-back** utility selects the boolean set operations used to update the 3D wafer geometry after a vertical etching or deposition. For etching update, the utility invokes the **subtraction** operation. For deposition update, the utility invokes either the **merge** operation, if the same material as the top layer material was deposited, or the **glue** operation, if a different material was deposited. As an example, the lower right corner of Figure 6.2 displays the two-holes structure after the **vertical deformation volume stitch-back** utility had updated the vertical deposition of the polysilicon layer.

In Figure 6.2, the number of faces or triangles is shown for each solid model or surface mesh. Using the same initial structure boundary meshing parameters, the number of faces or triangles may vary slightly for other TCAD systems or geometry servers. There are two sources for such discrepancies. First, the **initial structure construction** utility does not specify how rectangular layer boundaries are tiled. For example, in the BTU system, a rectangular layer's *y* direction divisions are created by extruding a rectangle parallel to the *xz*-plane. Consequently, as shown in the upper left corner of Figure 6.2, there are **NY** rectangular panels on each of the layer boundaries at x = xmin, x = xmax, and z = zmin. Other TCAD systems may easily choose to create one large rectangle in place of **NY** rectangular panels. Secondly, boolean set operation implementations tend to vary greatly in how they compute solid intersection curves, and split existing faces from these curves. Therefore, especially in the construction of two-holes initial structures, where cone stubs are subtracted from rectangular layers, different geometry servers may create different numbers of faces near the holes.

Figure 6.1

Cross Sectional Geometry Dimensions and Positions for Planar Stack and Two–Holes Initial Structures

Cross Section of Two–Holes Structure at Y = 0



IBM Geometry Engine Structures Created By BTU Standardized Performance Testing Utilities

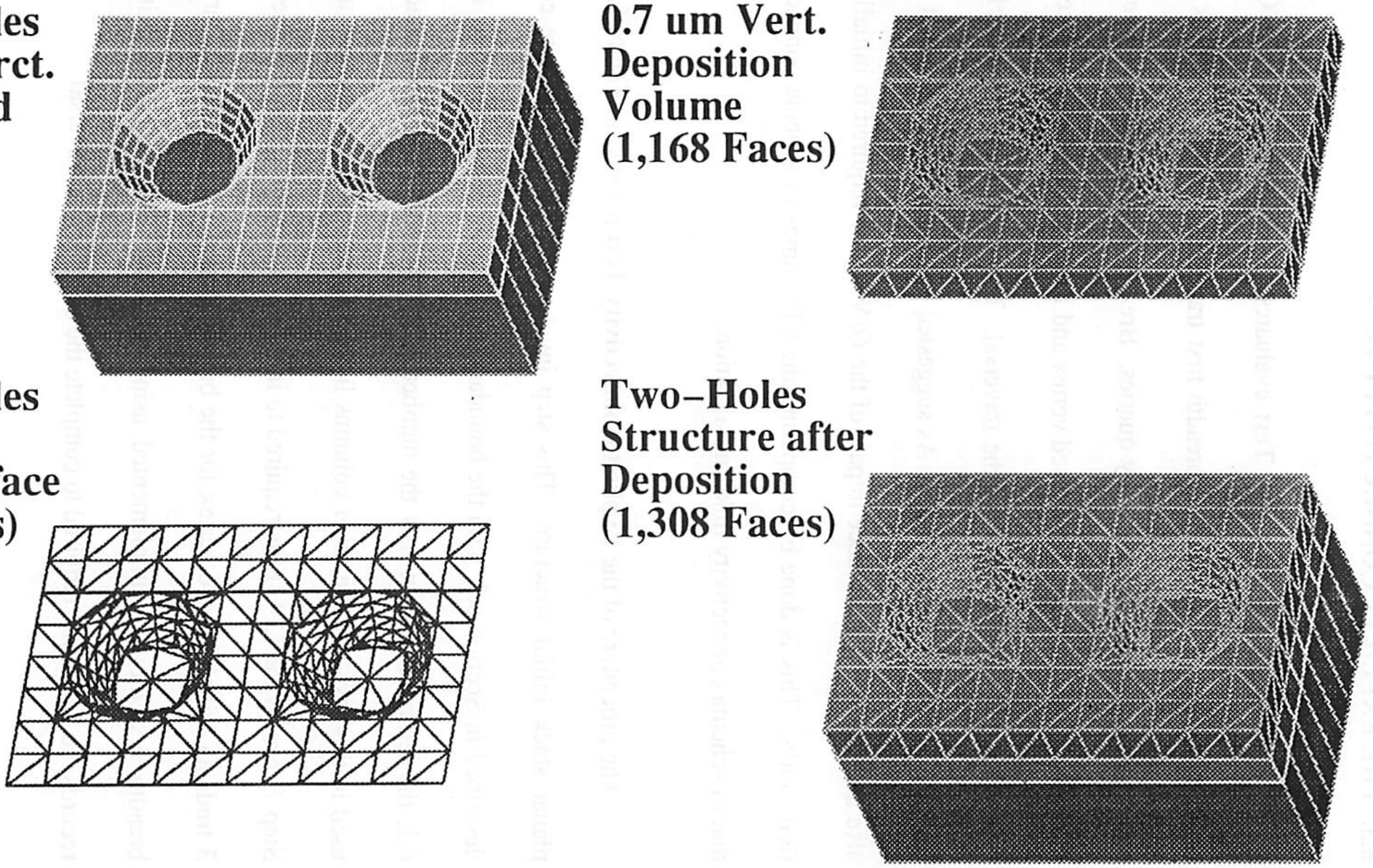


Figure 6.2

RHW – UCB TCAD

6.3. THE EXPLICIT CONNECTIVITY TEST

The **Explicit Connectivity Test** evaluates the efficiency of IBM Geometry Engine's **star-edge schema**. In this test, breadth first traversal of surface vertices is used to provide representative surface connectivity queries. Breadth first traversal involves recursively using the connectivity between a traversed vertex and its adjacent edges and faces to find connected vertex neighbors for continuing the traversal. Therefore, this traversal effectively exercises useful schema connectivity links. As suggested in Chapter 5, the **Explicit Connectivity Test** also assesses the performance impact of the $O(N^2)$ time required to initialize schema connectivity links. This is done by comparing the CPU times of schema connectivity queries with that of schema connectivity links initialization.

The procedure of the **Explicit Connectivity Test** is listed in Figure 6.3. Step 1 creates a planar stack initial structure. This step invokes the **initial structure construction** utility described in Section 6.2, with the boundary meshing parameters listed in Table 6.1. In Table 6.1, the leftmost column lists the number of solid faces in the planar stack initial structures used in this test, and the other columns list the corresponding boundary meshing parameters. Step 2 records the CPU time required to initialize **star-edge schema** connectivity links. Steps 3 finds an initial surface vertex for the breadth first traversal. Steps 4 through 6 describes a breadth first traversal implemented using **star-edge schema** connectivity queries. Step 7 records the CPU time required to complete the breadth first traversal.

As shown in Table 6.1, the **Explicit Connectivity Test** was conducted on planar stack initial structures with 72 to 496 faces. Figure 6.4 depicts an example of the IBM Geometry Engine solid structures created for this test. The left of Figure 6.4 plots the wireframe model of a planar stack initial structure with 72 faces. The right of Figure 6.4 shows the surface vertices reported during the breadth first traversal.

Results of the **Explicit Connectivity Test** confirmed that the IBM Geometry Engine's **star-edge schema** connectivity links can be efficiently used to find connected vertex neighbors in *O(1) (constant)* time. On the lower curve, Figure 6.5 plots the CPU times used to perform breadth first traversals, versus the number of solid faces. In Figure 6.5, CPU times are plotted along the *y*-axis, and the number of solid faces are plotted along the *x*-axis. As shown in Figure 6.5, breadth first traversal of surface vertices required only *O(N)* time. For IBM Geometry Engine solids with 70 faces to 500 faces, went from about 0.5 seconds to about 5 seconds. In a solid with N faces, there are *O(N)* surface vertices. Therefore, test results confirmed that at each surface vertex, only *O(1) constant* time is used to find the small number of connected vertex neighbors.

For 3D IC topography simulation, **Explicit Connectivity Test** results suggested that the efficiency gained in using the **star-edge schema** may be negated by the cost of initializing schema connectivity links. On the upper curve, Figure 6.5 plots the CPU times used to initialize schema connectivity links, versus the number of solid faces. As shown in Figure 6.5, initialization of schema connectivity links required $O(N^2)$ time. For planar topographies with 70 faces to 500 faces, test results showed that initialization of schema connectivity links went from about 2 seconds to about 100 seconds. These results also showed that initialization of schema connectivity links was from 4x to 20x more expensive than breadth first traversal of surface vertices.

Explicit Connectivity Test

Procedure:

1. Create a planar stack initial structure (see Table 6.1).

2. Record the CPU time required to initialize schema connectivity links.

3. Find and mark the surface vertex whose (x,y) are nearest to (x center, y center) of stack structure.

4. Find all surface faces incident on this surface vertex.

5. For each incident surface face, find all connected vertex neighbors.

6. For each unvisited vertex neighbor, Repeat Steps 4 to 6.

7. Record the CPU time used in breadth first traversal.

Figure 6.3

RHW – UCB TCAD

Table 6.1

Explicit Connectivity Test
Boundary Meshing Parameters

X Length = 6 um, Y Length = 4 um

# of Layers = 2

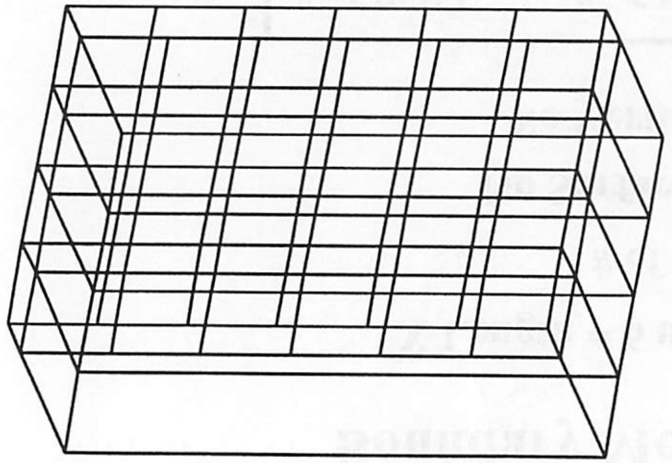
No Surface Triangulation.

No Vertical Deposition.

# of Solid Faces (N)	# of X Spaces per um (NX)	# of Y Spaces per um (NY)
72	1	1
236	2	2
496	3	3

Geometry Engine Structures for Explicit Connectivity Test

Planar Stack
(72 Faces)



Planar Stack with
Traversed Vertices

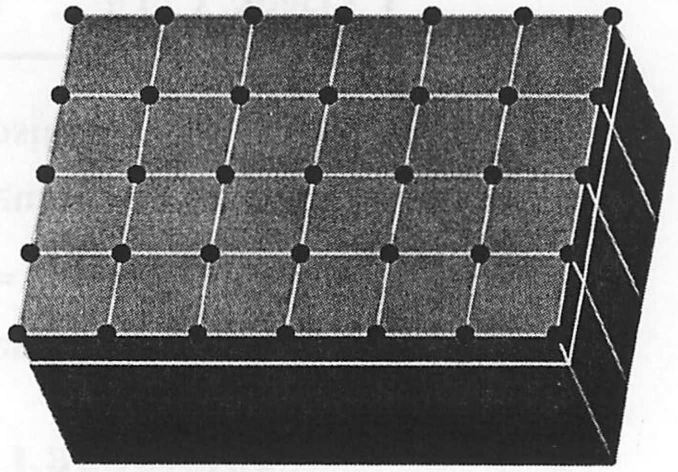


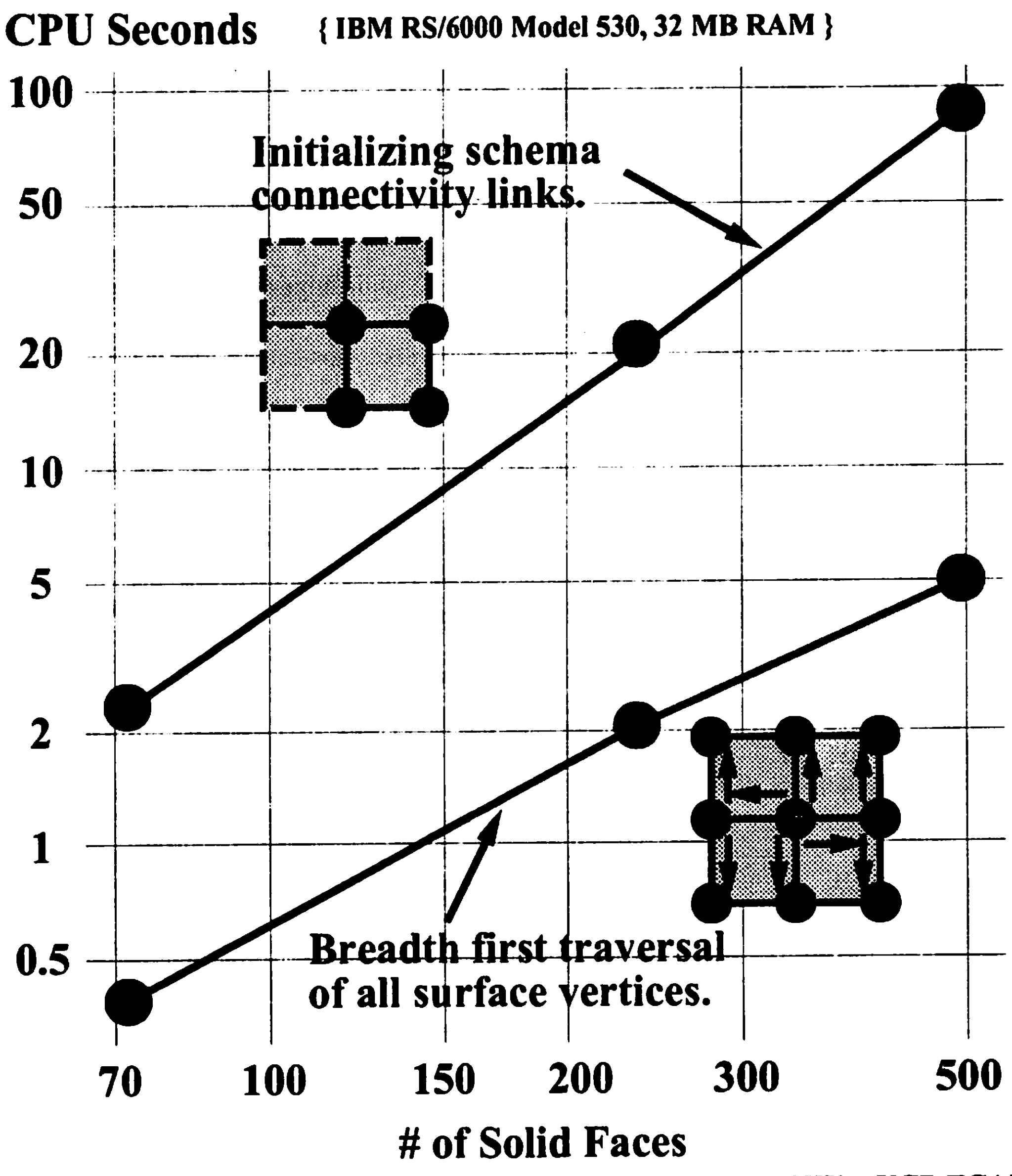
Figure 6.4

RHW – UCB TCAD

Explicit Connectivity Test

Explicit connectivity exists in the start–edge schema. –> Breadth first traversal in O(N) time.

Initializing schema connectivity links requires O(N^2) time, and at least 10x the CPU times for breadth first traversal.



RHW – UCB TCAD

Figure 6.5

6.4. THE FACE-FACE INTERSECTION SORTING TEST

The **Face-Face Intersection Sorting Test** demonstrates the efficiency gained through IBM Geometry Engine's **bucket sorting of face bounding boxes** data structure. This test uses boolean set operations to glue 3D IC topographies with their vertical deposition volumes. In this test, face-face intersections are computed between the IC topographies and their vertical deposition volumes. An important feature of the test is that it introduces 1) *O(N)* deposition volume triangles that lie exactly on initial surface faces, and 2) *O(N)* deposition volume triangles that lie away from initial surface faces. As suggested in Chapter 5, the **Face-Face Intersection Sorting Test** also characterizes the performance impact of using $O(N^2)$ face bounding box intersection computations to compute face-face intersections between planar topographies and their vertical deposition volumes. This is done by comparing the CPU times used to compute solid intersection curves on planar versus non-planar topographies.

The procedure of the **Face-Face Intersection Sorting Test** is listed in Figure 6.6. Step 1 constructs a planar stack or two-holes initial structure with a 6 um thick silicon substrate. This choice of substrate thickness forces the creation of face bounding boxes along the *z*-axis. This step invokes the **initial structure construction** utility described in Section 6.2, with the boundary meshing parameters listed in Table 6.2. Table 6.2 lists the parameters for the planar stack initial structures in the top five rows, and those for the two-holes initial structures in the bottom four rows. In Table 6.2, the leftmost column lists the number of surface triangles spawned by the initial structures, and the other columns list the corresponding boundary meshing parameters. Step 2 creates a 0.7 um polysilicon vertical deposition volume using the **vertical deformation volume construction** utility described in Section 6.2. Step 3 stitches the vertical deposition volume onto the initial structure using the **vertical deformation volume stitch-back** utility described in Section 6.2. Step 4 records the CPU time for the sub-task of computing the solid intersection curve.

As shown in Table 6.2, the **Face-Face Intersection Sorting Test** was conducted on planar stack initial structures that spawned 96 to 1,200 surface triangles, and two-holes initial structures that spawned 112 to 1,136 surface triangles. Figure 6.7 depicts examples of the initial structures and their vertical deposition volumes. The left of Figure 6.7 illustrates a planar stack initial structure that spawned 192 surface triangles. The right of Figure 6.7 illustrates a two-holes initial structure that spawned 320 surface triangles.

Results of the **Face-Face Intersection Sorting Test** confirmed that the IBM Geometry Engine's **bucket sorting of face bounding boxes** data structure can be efficiently used to compute all face-face intersections between IC topographies and their vertical deposition volumes in expected $O(N)$ time. On the dashed curve, Figure 6.8 plots the CPU times used to compute solid intersection curves between two-holes initial structures and their vertical deposition volumes, versus the number of surface triangles in the structures. In Figure 6.8, CPU times are plotted along the y-axis, and the number of surface triangles are plotted along the x-axis. For non-planar topographies with 100 to 1,000 surface triangles, CPU times used by solid intersection computation went from about 15 CPU seconds to about 125 CPU seconds.

For 3D IC topography simulation, **Face-Face Intersection Sorting Test** results showed that there is minimal performance degradation due to the invocation of $O(N^2)$ face bounding box intersection computations in the planar topography case. On the solid curve, Figure 6.8 plots the CPU times used to compute solid intersection curves between planar stack initial structures and their vertical deposition volumes, versus the number of surface triangles in the structures. Compared to the non-planar topography case, test results for the planar topography case showed about 30% performance degradation in solid intersection computation time. For planar topographies with 100 to 1,000 surface triangles, CPU times used by solid intersection computation went from about 12 CPU seconds to about 150 CPU seconds.

Face–Face Intersection Sorting Test

Procedure:

1. Create a planar stack or two–holes initial structure (see Table 6.2).

2. Create a 0.7 um vertical deposition volume of the initial structure.

3. Stitch the 0.7 um vertical deposition volume to the initial structure.

4. Record the CPU time required to compute solid intersection curves between the initial structure and its vertical deposition volume.

Figure 6.6

RHW – UCB TCAD

Table 6.2

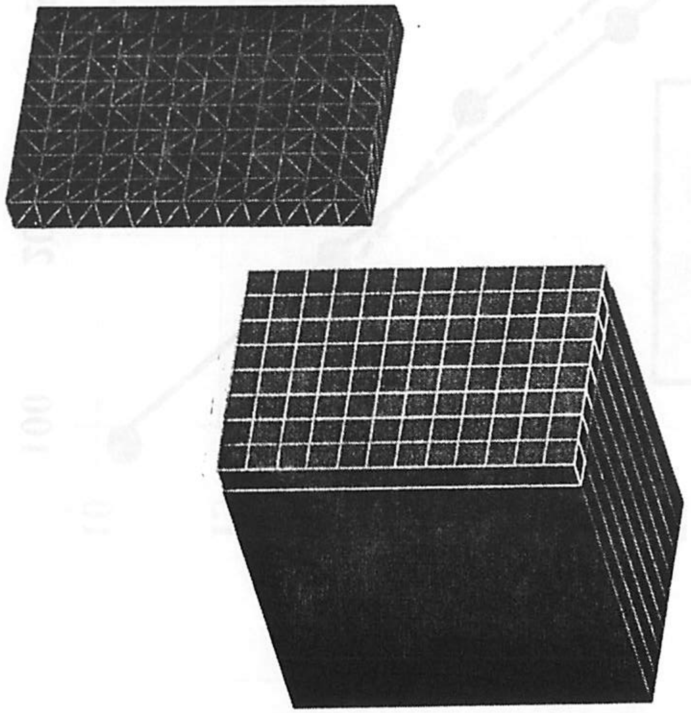
Face–Face Intersection Sorting Test
Initial Surface Meshing Parameters

X Length = 6 um, Y Length = 4 um, Z Substrate = 6 um, # of Layers = 2

# of Surface Triangles (N)	# of X Spaces per um (NX)	# of Y Spaces per um (NY)	# of Angular Divisions (NTheta)	# of Slices (NSlices)
96	1	2		
192	2	2		
432	3	3		
768	4	4		
1,200	5	5		
112	1	2	4	1
320	2	2	8	2
664	3	3	12	3
1,136	4	4	16	4

Geometry Engine Structures for Face–Face Intersection Sorting Test

Planar Stack Structure + Deformation Volume (192 Surface Triangles)



Two–Holes Structure + Deformation Volume (320 Surface Triangles)

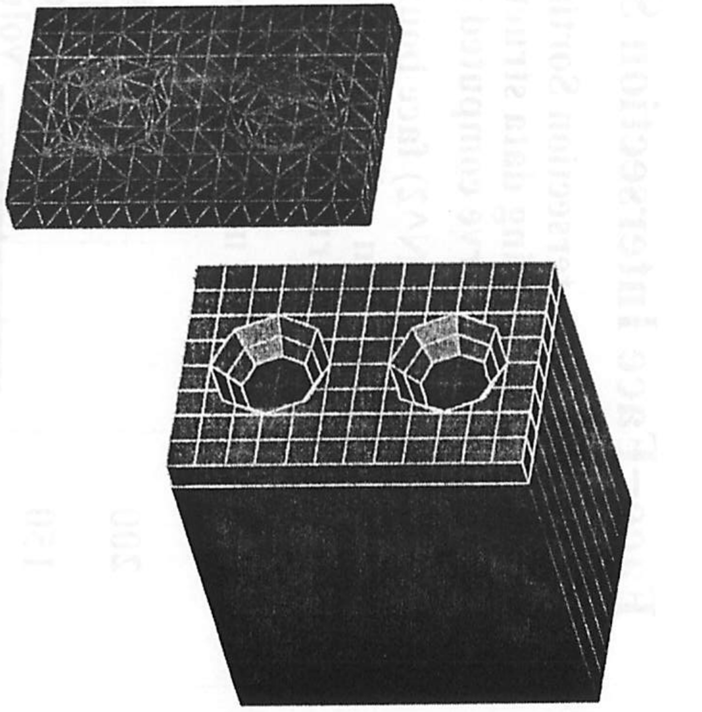


Figure 6.7

Face–Face Intersection Sorting Test

Face–Face Intersection Sorting using the bucket sorting data structure. –> Solid intersection curve computed in O(N) time.

Issuing O(N^2) face bounding box intersection calls results in about 30% performance degradation.

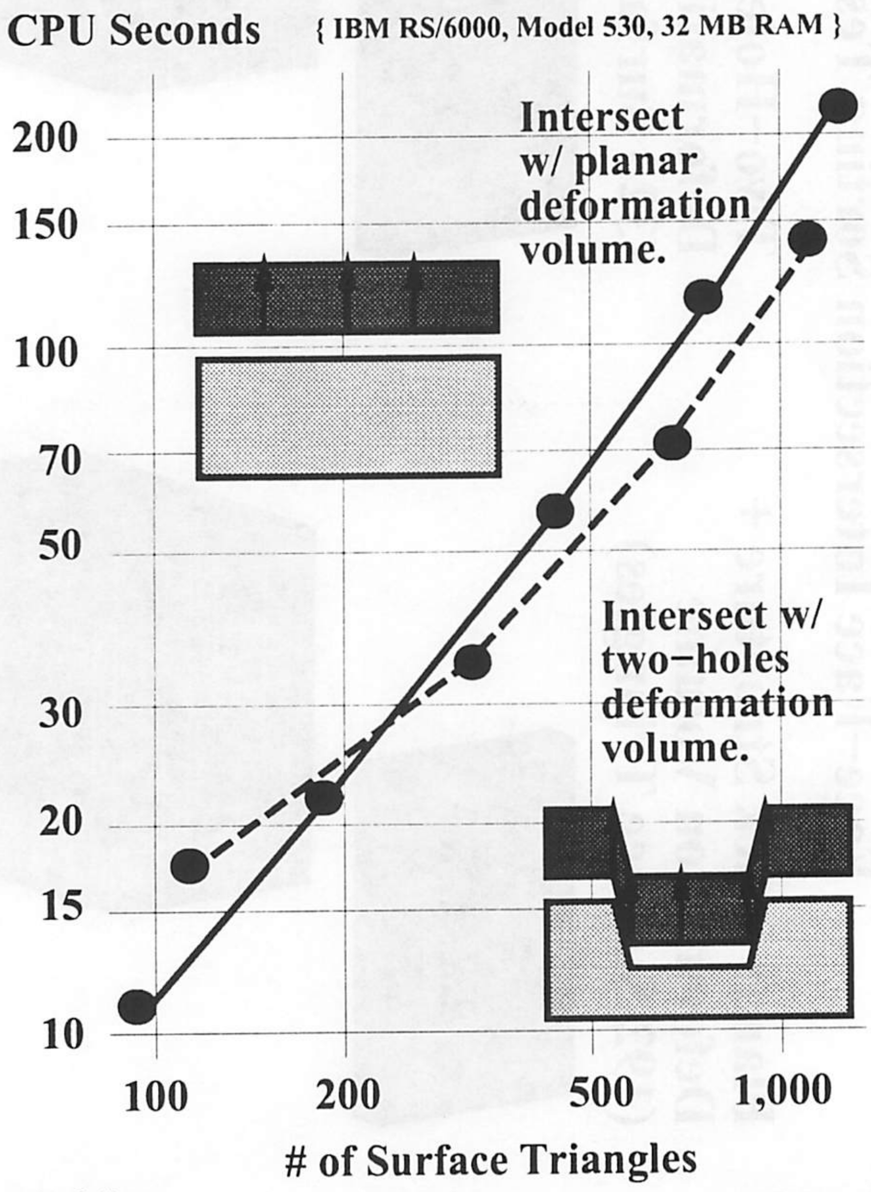


Figure 6.8

RHW – UCB TCAD

6.5. THE RAY-FACE INTERSECTION SORTING TEST

The **Ray-Face Intersection Sorting Test** has two parts. The first part (Part A) evaluates the performance impact of not supporting Ray-Face Intersection Sorting in the IBM Geometry Engine. In this part of the test, material interace collision detection is emulated by applying **point location tests** to points on a vertical etch front. In the IBM Geometry Engine, the **point location test** is implemented by shooting out a ray from a query point towards the *-x* direction, and using ray-face intersections to identify the vertex, edge, face, or volume that contains the query point. The **point location test** terminates as soon as it obtains a sufficient number of ray-face intersections to determine the containing geometry component. In other words, the **point location test** does not necessarily find all the faces hit by the ray.

The second part (Part B) of the **Ray-Face Intersection Sorting Test** demonstrates the performance improvement that could be gained by using the **bucket sorting of face bounding boxes** data structure to support Ray-Face Intersection Sorting. Another purpose of this part of the test is to introduce **geometrical operation transformation** as a data organization method to compensate for the lack of Ray-Face Intersection Sorting. In this part of the test, **boolean set operations** between sliver teterahedra and 3D topographies, replace **point location tests** on etch front points. Each sliver tetrahedron consists of an apex located near the center of the topography, and a vertical triangular base located to the left of the topography.

The procedure of the **Ray-Face Intersection Sorting Test** is listed in Figure 6.9. Step 1 constructs a planar stack or two-holes initial structure. This step invokes the **initial structure construction** utility described in Section 6.2, with the initial boundary meshing parameters listed in Table 6.3. Table 6.3 lists the initial parameters for planar stack vertically deposited structures in the top four rows, and those for two-holes vertically deposited structures in the bottom four rows. In Table 6.3, the leftmost column lists the number of faces in the vertically deposited structures (i.e. solid structures produced by Step 2 and 3), and the other columns list the corresponding initial boundary meshing parameters. Steps 2 and 3 in this test are identical to Steps 2 and 3 in the **Face-Face Intersection Sorting Test** described in Section 6.3.

The remaining procedure for the first part (Part A) of the **Ray-Face Intersection Sorting Test** is listed in the upper portion of Figure 6.9. Steps 4a creates a set of fake etch points at -0.1 below current surface vertices. In Step 4a, one fake etch point is created for each surface vertex. For each fake etch point, Step 5a finds, the vertex, edge, face, or volume that contains it. Step 6a records the total CPU time used to locate all points. Step 7a calculates the average CPU time by dividing the total CPU time by the number of fake etch points.

The remaining procedure for the second part (Part B) of the **Ray-Face Intersection Sorting Test** is listed in the lower portion of Figure 6.9. Step 4b creates a sliver tetrahedron with an apex at (0,0, ztop - 0.1) (i.e. a representative fake etch point), and a small triangular base at x = xmin - 0.1. Step 5b intersects the initial structure with this sliver tetrahedron. Step 6b records the CPU times used to compute the solid intersection curve, and perform the subtask of bucket sorting face bounding boxes. As shown in Figure 6.9, this procedure implements a **geometrical operation transformation** by replacing a **point location test** with a **boolean set operation.**

As shown in Table 6.3, the **Ray-Face Intersection Sorting Test** was conducted on planar stack vertically deposited structures with 340 faces to 2,164 faces, and two-holes vertically deposited structures with 312 faces to 2,876 faces. Figure 6.10 depicts examples of these structures, and the sliver tetrahedra they intersected with. The left of Figure 6.10 shows a planar stack vertically deposited structure with 604 faces. The right of Figure 6.10 shows a two-holes vertically deposited structure with 860 faces.

Results from the first part of the the **Ray-Face Intersection Sorting Test** confirmed that, due to the lack of Ray-Face Intersection Sorting, IBM Geometry Engine **point location test** runs in *O(N)* time. On the dashed curve, Figure 6.11 plots the average CPU times used by each **point location test** on two-holes vertically deposited structures. In Figure 6.11, CPU times are plotted along the *y*-axis, and the number of solid faces are plotted along the *x*-axis. For two-holes deposited structures with 300 faces to 3,000 faces, average CPU times for each **point location test** went from about 0.1 second to about 1 second.

As further confirmation, **point location tests** were also performed on planar stack vertically deposited structures with comparable numbers of solid faces. On the solid curve, Figure 6.11 plots the average CPU times on the planar stack deposited structures. For planar stack deposited structures with 350 faces to 2,100 faces, average CPU times went from about 0.11 seconds to about 0.60 seconds. As shown in Figure 6.11, average CPU times per **point location test** for planar stack deposited structures were consistently about 30% less than those for two-holes deposited structures. This is because each ray would only need to hit one face to determine the volume that contains the point.

Results from the second part of the the **Ray-Face Intersection Sorting Test** showed that the **bucket sorting of face bounding box** data structure can be used to find ray-face intersections in strongly sublinear time for both non-planar or planar topographies. Moreover, by bucket sorting topographies with at least 2,000 faces, CPU times per sliver intersection can be at least 10x less than CPU times per **point location test**. On the dashed curve, Figure 6.12 plots the CPU times per sliver intersection on two-holes deposited structures. On the solid curve, Figure 6.12 plots the CPU times per sliver intersection on planar stack deposited structures. In Figure 6.12, CPU times are plotted along the *y*-axis, and the number of solid faces are plotted along the *x*-axis. For a two-holes deposited structure with about 2,000 faces, CPU time per sliver intersection was about 0.08 seconds, while CPU time per **point location test** was about 0.8 seconds. For a planar stack deposited structure with about 2,000 faces, CPU time per sliver intersection was about 0.03 seconds, while CPU time per **point location test** was about 0.6 seconds.

Ray–Face Intersection Sorting Test

Procedure:

1. Create a planar stack or two–holes initial structure (see Table 6.3).

2 and 3. Same as Steps 2 and 3 in the Face–Face Intersection Sorting Test.

A: Use Ray–Face Intersection.

4a. Create points at −0.1 um below current surface vertices (i.e. one point for each vertex).

5a. For each fake etch point, find the vertex, edge, face, or material volume that contains it (i.e. use ray shooting).

6a. Record the total CPU time used to locate all points.

7a. Divide total CPU time by number of points (varies).

B: Use Face–Face Intersection with Slivers.

4b. Create a sliver tetrahedron with (0,0, ztop − 0.1), (xmin − 0.1, 0.1, −0.1), (xmin − 0.1, 0, −0.2), (xmin − 0.1, −0.1, −0.1).

5b. Intersect initial structure with sliver tetrahedron.

6b. Record the CPU time used to compute solid intersection, and the sub–task of bucket sorting face bounding boxes.

Figure 6.9

Table 6.3

Ray–Face Intersection Sorting Test
Initial Boundary Meshing Parameters

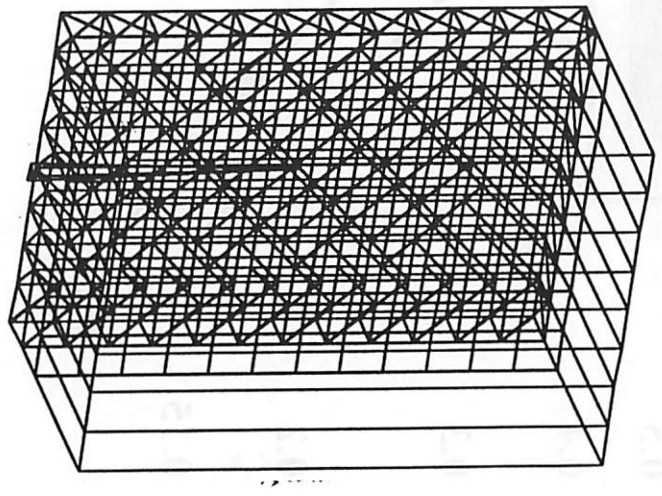
X Length = 6 um, Y Length = 4 um
# of Layers = 3 (Vertical Deposition)

# of Face: Vert. Depo. Solid (N)	# of X Spaces per um (NX)	# of Y Spaces per um (NY)	# of Angular Divisions (NTheta)	# of Slices (NSlices)
340	1	2		
604	2	2		
1,264	3	3		
2,164	4	4		
312	1	1	4	1
860	2	2	8	2
1,724	3	3	12	3
2,876	4	4	16	4

Geometry Engine Structures for Ray–Face Intersection Sorting Test

Planar Stack Structure w/ 0.5 um v. deposited poly layer (604 faces) + "–x" dir. ray prism

Two–Holes Structure w/ 0.5 um v. deposited poly layer (860 faces) + "–x" dir. ray prism



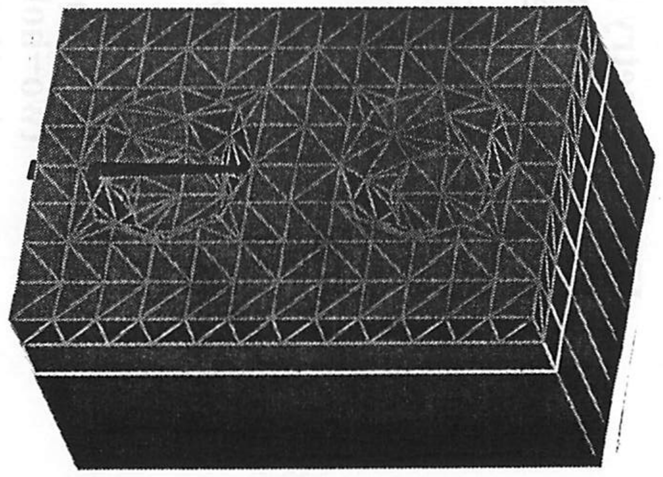


Figure 6.10

RHW – UCB

Ray−Face Intersection Sorting Test − 1

IBM Geometry Engine does not have
Ray−Face Intersection Sorting. ->
Point location test is O(N) time.

Point location test terminates as soon
as containing component is found. ->
Time grows as 5x from 200 to 2,000 faces,
and planar case is faster (i.e.fewer hits).

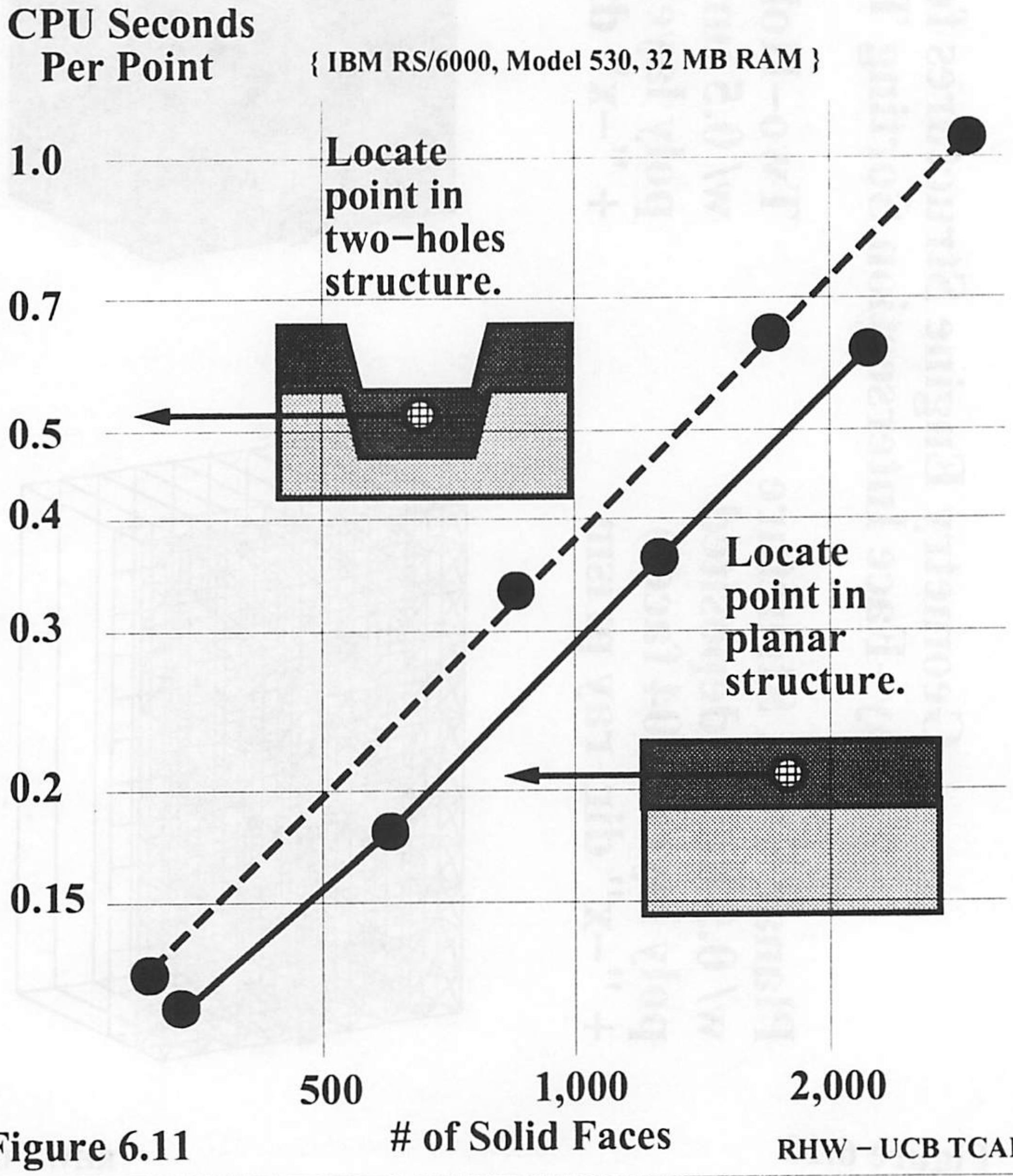


Figure 6.11

RHW − UCB TCAD

Ray–Face Intersection Sorting Test – 2

{ Note: Does not include preprocessing time! }

Replace ray–face intersections with face–face intersections. –> Gain at least 10x CPU time reduction per point for 2,000+ faces.

"Point location" faster in planar case due to fewer face–face intersections.

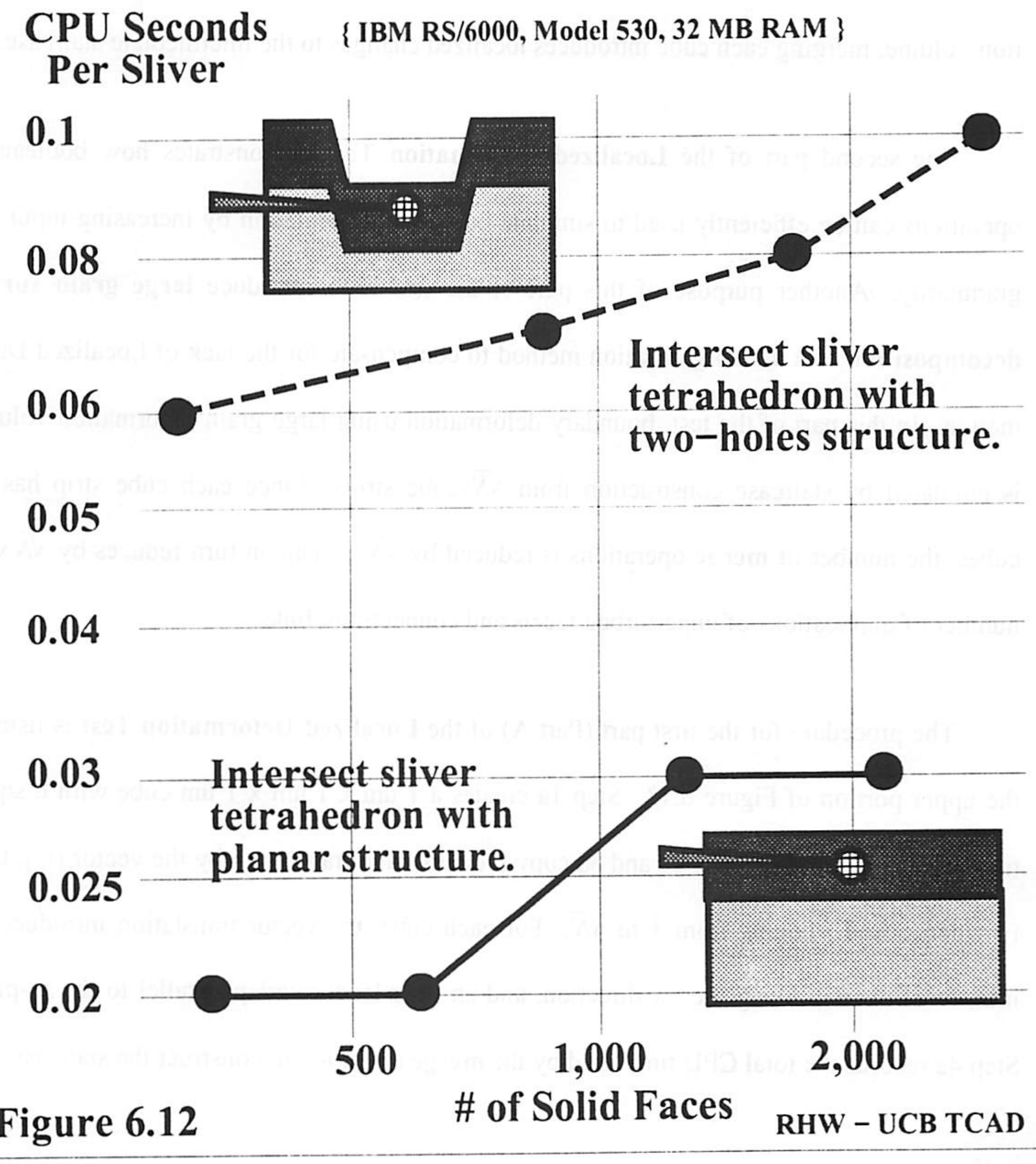


Figure 6.12

RHW – UCB TCAD

6.6. THE LOCALIZED DEFORMATION TEST

The **Localized Deformation Test** consists of two parts. The first part (Part A) demonstrates the performance impact of not supporting Localized Deformation in the IBM Geometry Engine. In this part of the test, boundary deformation using triangular prisms is emulated by staircase construction from cubes. In the context of boundary deformation, a cube is computationally equivalent to a triangular prism generated by a triangular facet sweep. Just as merging each triangular prism introduces localized changes to the intermediate aggregate deformation volume, merging each cube introduces localized changes to the intermediate staircase.

The second part of the **Localized Deformation Test** demonstrates how boolean set operations can be efficiently used to simulate boundary deformation by increasing input data granularity. Another purpose of this part of the test is to introduce **large grain surface decomposition** as a data organization method to compensate for the lack of Localized Deformation. In this part of the test, boundary deformation using large grain deformation volumes is emulated by staircase construction from $\sqrt{N}$-cube strips. Since each cube strip has $\sqrt{N}$ cubes, the number of **merge** operations is reduced by $\sqrt{N}$x. This in turn reduces by $\sqrt{N}$x the number of duplications of unperturbed facets and connectivity links.

The procedure for the first part (Part A) of the **Localized Deformation Test** is listed in the upper portion of Figure 6.13. Step 1a creates a 1 um x 1 um x 1 um cube with 6 square tiles. For N iterations, Steps 2a and 3a copy the cube, and translate it by the vector (i, j, 0.5 * i), where i and j counts from 1 to $\sqrt{N}$. For each cube, the vector translation introduces an intra-level overlap along the *+x* direction, and an inter-level overlap parallel to the *xz*-plane. Step 4a records the total CPU time used by the **merge** operation to construct the staircase.

The procedure for the second part (Part B) of the **Localized Deformation Test** is listed in the lower portion of Figure 6.13. Step 1b creates a 1 um x $\sqrt{N}$ um x 1 um cube strip with $\sqrt{N}$ tiles along the $+y$ direction. For $\sqrt{N}$ iterations, Step 2b copies the $\sqrt{N}$ cube strip, and translates it by the vector (0, 0.5 * i, i), where i counts from 1 to $\sqrt{N}$. For each $\sqrt{N}$ cube strip, the vector translation introduces an inter-level overlap parallel to the xy-plane. Step 3b records the total CPU time used by the **merge** operation to construct the staircase.

The **Localized Deformation Test** was conducted for staircases with 100, 400, 900, and 1600 cubes. Figure 6.14 illustrates examples of staircases constructed using cubes and $\sqrt{N}$ cube strips. The left of Figure 6.14 depicts a staircase created by merging 100 cubes. The right of Figure 6.14 depicts an equivalent staircase created by merging 10 10-cube strips.

Results from the first part of the **Localized Deformation Test** confirmed that, due to the lack of Localized Deformation, staircases can be merged in worst case $\frac{1}{2} \times O(N^2 logN)$ time. On the upper curve, Figure 6.15 plots the total CPU times used by IBM Geometry Engine **merge** operation to construct staircases with 100 to 1,600 cubes. In Figure 6.15, CPU times are plotted along the y-axis, and numbers of staircase cubes are plotted along the x-axis. For staircases with 100 to 1,600 cubes, total CPU time used by the **merge** operation went from about 120 seconds to about 20,000 seconds. In other words, a 16x increase in total number of staircase cubes translated to a 167x increase in total CPU time. More importantly, this result confirmed that brute force application of IBM Geometry Engine boolean set operations is clearly not a practical method for simulating boundary deformation.

Results from the second part of the **Localized Deformation Test** showed that increasing input data granularity led to a more *O(N)*-like growth in total CPU time used by IBM Geometry Engine **merge** operation. On the lower curve, Figure 6.15 plots the CPU time the total CPU times used by IBM Geometry Engine **merge** operation to construct staircases from 10 10-cube strips to 40 40-cube strips. As shown in Figure 6.15, the CPU time for merging 10 10-cube strips (N = 100) was about 15 seconds, while the CPU time for merging 40 40-cube strips (N = 1600) was about 500 seconds. Here, a 16x increase in total number of staircase cubes only resulted in a 33x increase in total CPU time. More importantly, for staircases with comparable numbers of cubes, increasing input data granularity by $\sqrt{N}$x consistently resulted in $\sqrt{N}$x reduction in total CPU time. For example, the CPU time for merging 1,600 cubes was about 20,000 seconds, while the CPU time for merging 40 40-cube stripes was about 500 seconds.

Localized Deformation Test

Procedure:

A: Staircase from N Cubes.

```
1a. Create a 1 um x 1 um x 1 um
    cube with 6 tiles.

2a. For i = 1 to sqrt(N),

3a.   For j = 1 to sqrt(N),
        Copy cube and
        translate by (i, j, 0.5 * i).

4a. Record the total CPU time
      used by merge operations
      to construct staircase.
```

B: Staircase from sqrt (N) sqrt (N)−cube strips.

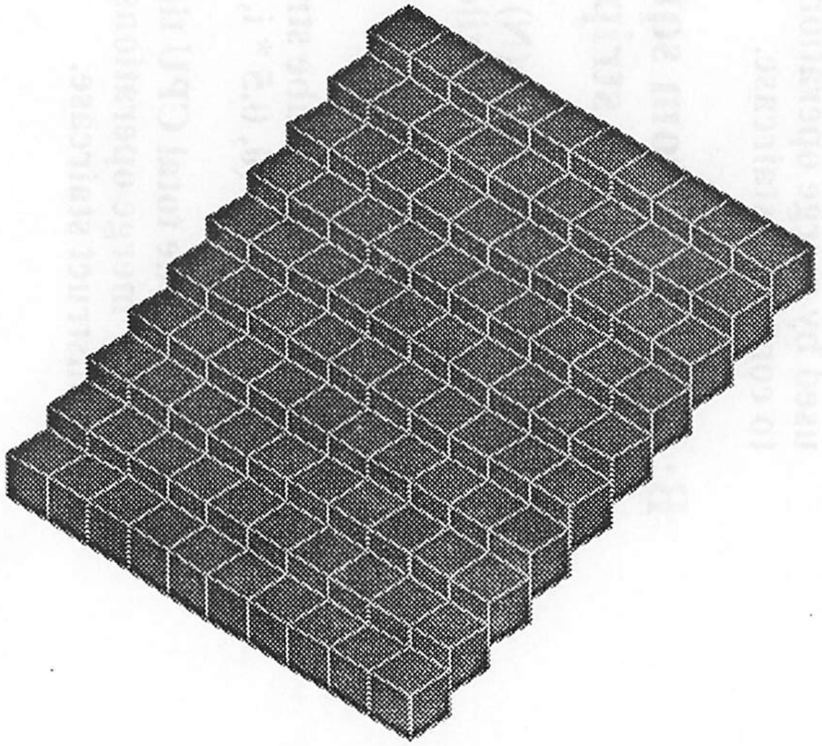
```
1b. Create a 1 um x sqrt(N) um x 1 um
    cube strip with sqrt(N) tiles along
    the +y direction.

2b. For i = 1 to sqrt(N)
        Copy sqrt(N)−cube strip and
        translate by (0, 0.5 * i, i).

3b. Record the total CPU time
      used by merge operations
      to construct staircase.
```

Figure 6.13

RHW – UCB TCAD

Geometry Engine Structures for Localized Deformation Test

Merge 100 Cubes



Merge 10 Slabs of 10 Cubes

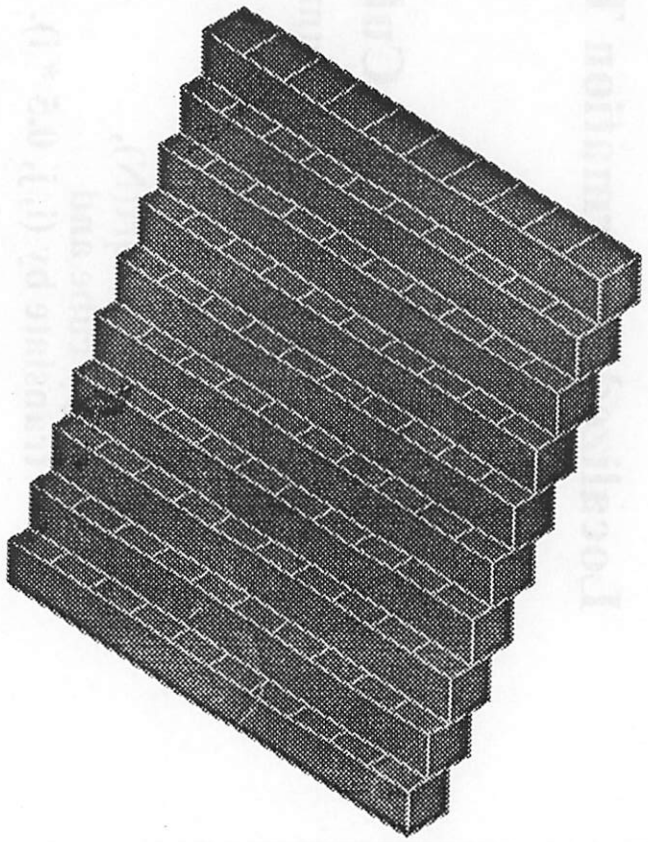


Figure 6.14

RHW – UCB TCAD

Localized Deformation Test

IBM Geometry Engine does not have Localized Deformation. -> Worst case total merge time (granularity = 1 cube) proportional to 0.5 * (N^2).

Total CPU time reduction factor = # of Operations reduction factor due to increased granularity.

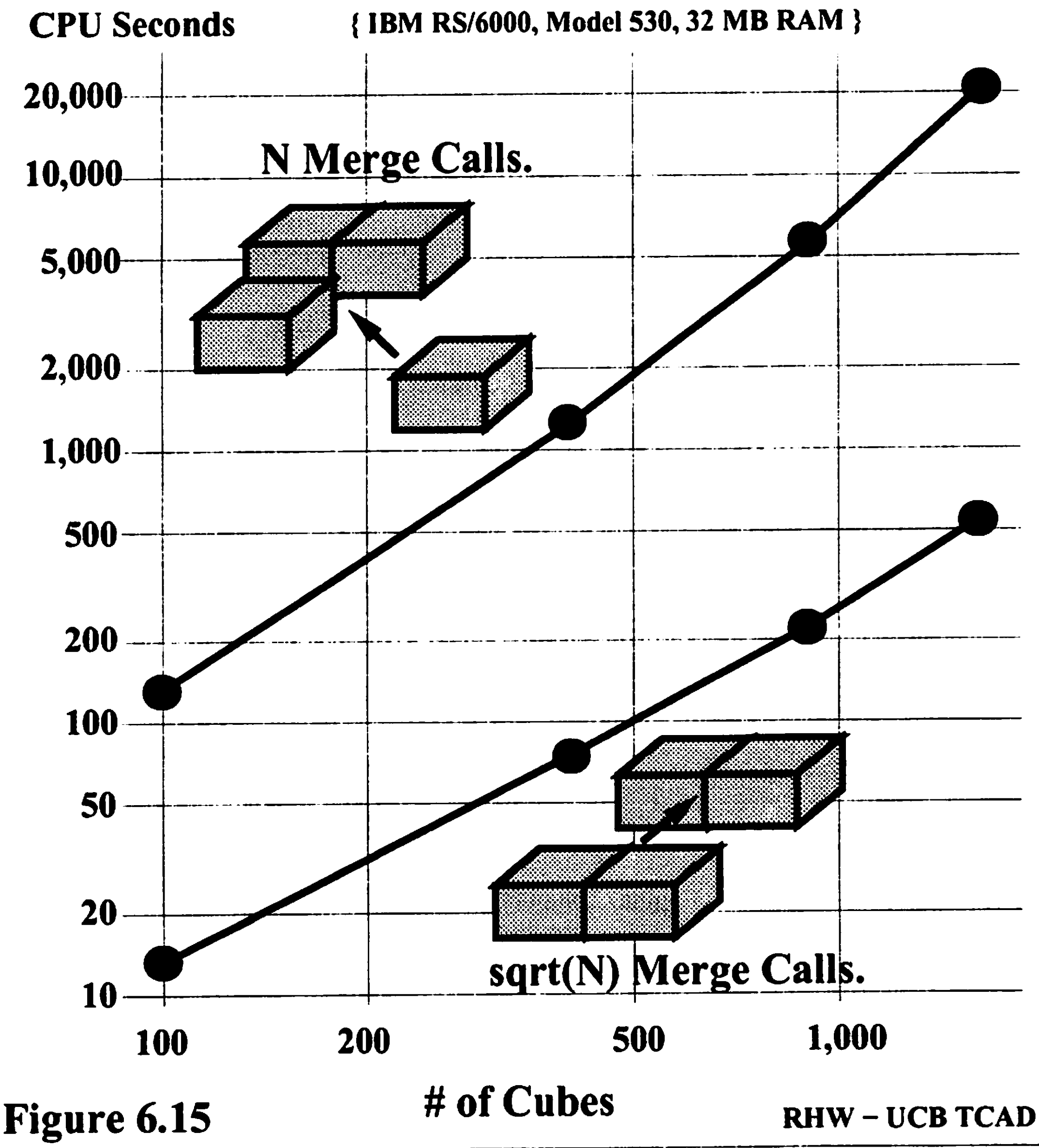


Figure 6.15

RHW - UCB TCAD

6.7. CONCLUSIONS

This chapter described **standardized performance tests** on the IBM Geometry Engine. These tests were designed to mimic the stress placed on geometry servers during 3D IC topography simulation. The tests invoke geometrical operations on IC topographies at typical levels of physical detail and operation frequency. In other words, standardized performance tests can be used to evaluate the suitability of geometry servers for 3D IC topography simulation.

In implementing standardized performance tests, it was shown in Section 6.2 that the demand of IC topography simulation required considerable augmentation of the standard interfaces to general-purpose geometry servers. About 10,000 lines of C++ code are likely necessary to perform relatively basic geometrical operations in 3D IC topography simulation with any solid modeling package. For example, about 12,000 lines of C++ code were required to interface the IBM Geometry Engine for performance testing. Out of these 12,000 lines, a significant part performs mundane geometry construction and data mapping tasks, such as constructing tiled initial topographies, extracting surface faces, triangulating polygonal faces, and constructing vertical deformation volumes.

Since standardized performance tests take into account the nature of geometrical operations, they are an indispensable system tool for characterizing the run time consequences of theoretical performance bounds. Standardized performance tests can screen out false performance bottlenecks often predicted from simpler asymtotic performance estimates. For example, in the IBM Geometry Engine, computing solid intersection curves between a planar topography and its vertical deposition volume results in a bucket with $O(N)$ triangles, and $O(N^2)$ face bounding box intersection checks. At first glance, for the special case of updating planar topographies, computing solid intersection curves appears to require $O(N^2)$ time.

However, performance test results in Section 6.4 showed the run time of solid intersection curve computation still grows as *O(N)* for planar topographies with 100 to 1,000 surface triangles. In fact, the $O(N^2)$ face bounding box intersection checks incurred in the planar case only cause about 30% run time performance degradation compared to the non-planar case.

Standardized performance tests can also reveal areas where geometry server design tradeoffs interact poorly with IC topography simulation needs. For robustness, most general-purpose boolean set operations create output solids by duplicating geometry components and connectivity links in the input solids. In Section 6.6, by means of a simple assembly of blocks into a staircase, it is possible to show that, for 1,600 cubes, an IBM Geometry Engine **merge** operation requires on the average about 10 seconds, regardless of whether the merge is a single block or a set of 40 cubes.

REFERENCES FOR CHAPTER 6

[1] M. Karasick, D. Lieber, Schemata for interrogating solid boundaries. ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[2] R.H. Wang, M.S. Karasick, and A.R. Neureuther, Computational evaluation of three-dimensional topography process simulation components. International Workshop on VLSI Process and Device Modeling (VPAD), Kyoto, Japan, May 1993, pp. 95-96. Robert VPAD reference.

CHAPTER 7

AUXILIARY DATA STRUCTURES FOR EFFICIENT USE OF GEOMETRY SERVERS

7.1. INTRODUCTION

This chapter introduces **monotone decomposition**, which was first published in [1][2], as an auxiliary data organization scheme to provide large grain surface decomposition for efficiently using geometry servers, such as the IBM Geometry Engine [3] and SAMPLE-3D [4]. In IC topography simulation, large numbers of locally connected facets often have similar orientations. By bin sorting locally connected facets with similar orientations, monotone decomposition can easily partition a simulated surface into large grain monotone patches. Using monotone decomposition, a surface advance with can be broken into a few well-behaved monotone patch advances, possibly with global inter-patch collisions. This can efficiently focus the power and robustness of merge operations in solid modelers to where it is most needed in IC topography simulation.

Section 7.2 reviews the feasibility of implementing these data organization methods as auxiliary data structures. As Section 7.2 will show, while geometrical operation transformation may require significant changes in geometry server internal data organization, large grain surface decomposition can be implemented independent of geometry server data representations. As an example of large grain surface decomposition, Section 7.2 introduces monotone decomposition as an effective decomposition scheme for IC topography simulation. Monotone decomposition groups locally connected facets which have similar facet orientations into large grain monotone patches.

Section 7.3 reviews some theoretical background for 2D monotone decomposition. Historically, the implementation of 2D monotone decomposition can be viewed as a simple extension of several classical concepts in 2D computational geometry, such as **monotone chains** and **monotone polygons** [5].

Two types of monotone decomposition are discussed. The first type, which is described in Sections 7.4 and 7.5, is Greedy monotone decomposition. **Greedy monotone decomposition** heuristically minimizes the number of geometrical operations, but may asymmetrically decompose axial symmetric IC topographical features, such as trenches. The 2D greedy monotone decomposition algorithm incrementally extends monotone chains going left to right along the top surface. The algorithm terminates a monotone chain when it reaches a surface facet that cannot monotonically extend the current chain.

Section 7.5 describes the **2.5D Isotropic Deposition Experiment** for evaluating greedy monotone decomposition. The test involves using IBM Geometry Engine **merge** operations to compute 2.5D boundary deformation of a 2D key hole trench. This test measures the reduction in number of **merge** operations and total CPU time due to the use of 2D greedy monotone decomposition.

The second type of monotone decomposition, which is described in Sections 7.6 through 7.8, is Directed monotone decomposition. By predefining monotone planes, **Directed monotone decomposition** can axially symmetrically decompose IC topographical features. The 3D directed monotone decomposition algorithm implicitly chooses monotone planes that partition facet orientations symmetrically about the local z-axes of IC topographical features. Section 7.6 shows that 3D directed monotone decomposition can use these facet orientation partitions to decompose complex topographical features into a few axial symmetric monotone patches.

Sections 7.7 and 7.8 describe two simulation experiments for evaluating directed monotone decomposition. Section 7.7 describes the **3D Isotropic Deposition Experiment**. This test is the full 3D equivalent of the **2.5D Isotropic Deposition Experiment**. The test shows that directed monotone decomposition makes it feasible to use IBM Geometry Engine **merge** operations in 3D boundary deformation.

Section 7.8 describes the **3D Source Visibility Experiment**. This test shows how directed monotone decomposition can be used to reduce the number of SAMPLE-3D **line-of-sight visibility tests** in computing 3D source visibility. It also demonstrates the need for accurate and efficient special-purpose shading interpolation algorithms that exploit facet spatial locality and orientation similarity inherent in monotone patches.

7.2. DATA ORGANIZATION METHODS FOR EFFICIENT USE OF GEOMETRY SERVERS

This section discusses the feasibility of implementing data organization methods as auxiliary data structures that improve the efficiency in using the IBM Geometry Engine. In Chapter 6, two data organization methods were introduced to compensate for missing constructs in the IBM Geometry Engine. The first method was geometrical operation transformation. This method involved replacing geometrical operations that were implemented without essential geometry server constructs, with similar operations that were implemented with essential constructs. The second method was large grain surface decomposition. This method increased input data granularity to reduce the number of CPU intensive primitive geometrical operations used to perform aggregate geometrical operations.

While **geometrical operation transformation** was previously shown to be effective at circumventing inefficient geometrical operations, implementing it as an auxiliary data structure may require significant changes in the internal geometry server data organization. In Section 6.5, IBM Geometry Engine **point location tests**, which had not been implemented with **ray-face intersection sorting**, were efficiently replaced by **boolean set operations**, which had been implemented with **face-face intersection sorting**. Implementing this geometrical operation transformation involves eliminating costly re-initializations of the **basket sorting of face bounding boxes** data structure during each boolean set operation. Figure 7.1 plots the run times for data structure initialization on the slivers and topographies used in the second part of the **Ray-Face Intersection Sorting Test**. As shown in Figure 7.1, data structure initialization times grew linearly as *O(N)*, and were comparable to the CPU times used by **point location tests**. To eliminate this inefficiency, the IBM Geometry Engine would need to be changed to make the bucket sorting data structure a part of the persistent server data.

On the other hand, **large grain surface decomposition** is an effective data organization method that can be efficiently implemented independent of geometry server data representations. Section 6.6 demonstrated the effectiveness of large grain surface decomposition to improve the efficiency in using IBM Geometry Engine **merge** operations for boundary deformation. By increasing the input data granularity by $\sqrt{N}$x, the staircase construction utility can reduce the number of IBM Geometry Engine **merge** operations and total CPU times by $\sqrt{N}$x. For any geometry server, large grain surface decomposition can be implemented using basic surface traversals to partition the simulated surface into large grain patches. These patches can be stored as auxiliary flags on geometry server data, or an alternative representation of the simulated surface.

This chapter describes **monotone decomposition** as an effective large grain surface decomposition auxiliary data structure for IC topography simulation. Monotone decomposition groups locally connected and similarly oriented facets into monotone patches. Due to the nature of IC processing technology, a typical simulated surface is a mostly planar surface that undergoes gradual and structured changes in facet positions and orientations. Therefore, at each time step, it should always be possible to decompose complex topographies into a few monotone patches. As the remainder of this chapter will demonstrate, efficient and innovative monotone decomposition algorithms can be implemented by combining basic surface traversals with fundamental computational geometry concepts.

Bucket Sorting DS Initialization Times: Ray−Face Intersection Sorting Test −2

Bucket Sorting DS initialization is O(N) time, and comparable to using point location tests.

In the IBM Geometry Engine, initialization is performed for each boolean set operation.

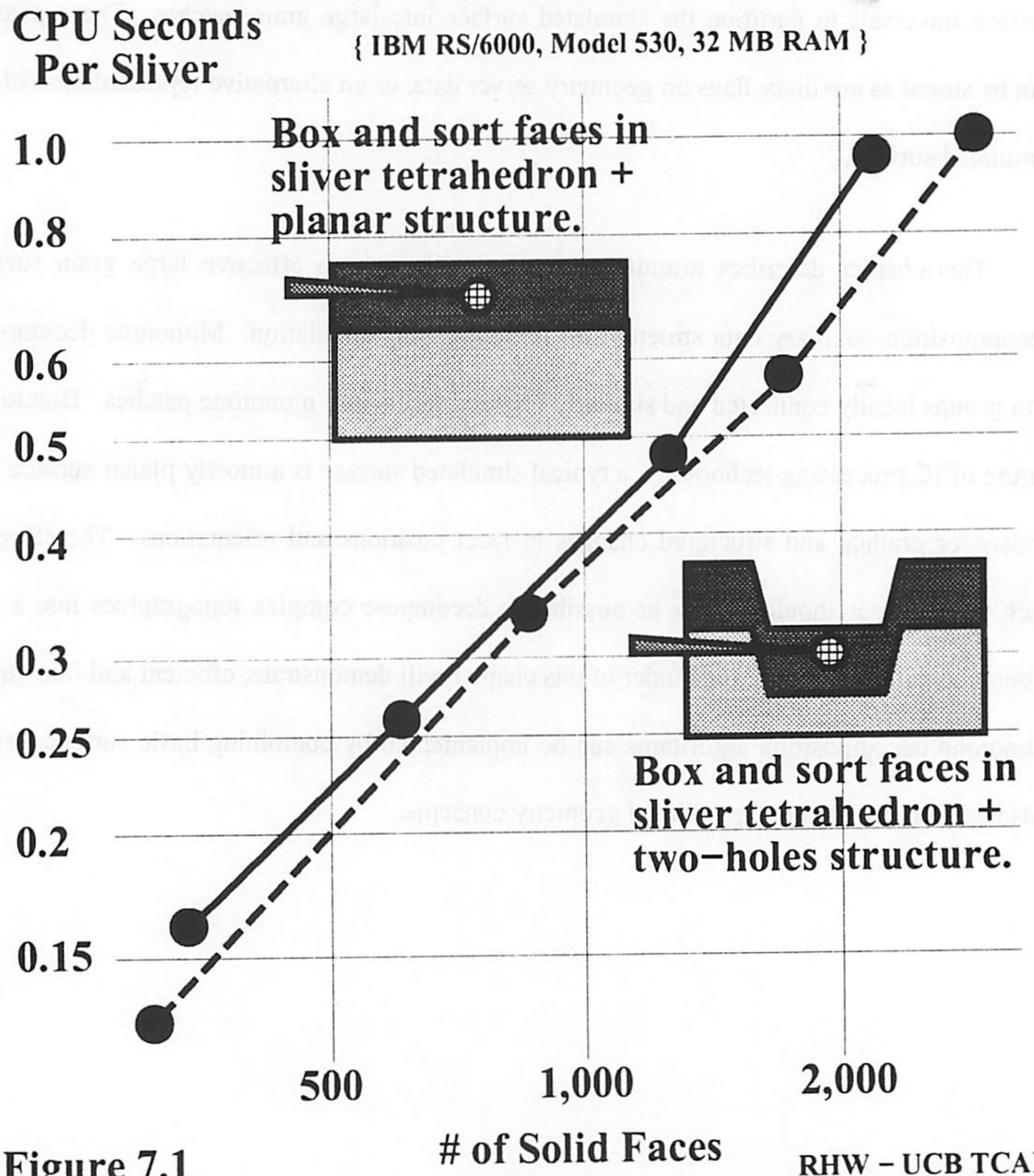


Figure 7.1

RHW − UCB TCAD

7.3. 2D MONOTONE CHAINS AND MONOTONE POLYGONS

This section formally defines classical computational geometry concepts, such as monotone chains and monotone polygons, which are relevant to implementing and using 2D monotone decomposition. Simply put, a monotone chain is an ordered sequence of locally connected facets that have similar orientations with respect to a straight line. A monotone polygon is a simple polygon which can be split into two chains that are monotone with respect to the same line. In the next two sections, these concepts will provide the necessary theoretical background for understanding the 2D greedy monotone decomposition algorithm and its application in efficiently using IBM Geometry Engine **merge** operations for 2.5D boundary deformation.

In 2D surface-based IC topography simulation, a chain is typically used to represent the topography top surface. A chain is an ordered sequence of vertices and directed facets. Figure 7.2 depicts a chain C with vertices {V1, ..., V10} and directed facets {F1 = (V1,V2), ..., F9 = (V9,V10)}. In this case, the chain C represents a symmetric trench top surface with slight overhangs and sloped sidewalls. As shown in Figure 7.2, Fi is a directed facet in the sense that, when Fi is traversed as Vi, Vi+1, air (i.e. outside) lies on Fi's left, and bounded material (i.e. inside) lies on Fi's right.

As defined by Preparata and Shamos [5], a **monotone chain** is a special type of chain with the following property:

A chain C = {V1, ..., Vn} is monotone with respect to a line L if any line orthogonal to L intersects C at exactly one point. (Definition 7.1)

This definition is illustrated in Figure 7.3, using a monotone chain C = {V1, ..., V5}, and its orthogonal projections on line L = {L(V1), ... L(V5)}. As shown in Figure 7.3, this definition offers no straightforward algorithm for decomposing an arbitrary chain into monotone chains.

On the other hand, the monotone chain definition does lead to the following useful property:

If chain C with vertices {V1, ..., Vn} is monotone with respect to line L, then the orthogonal projections {L(V1), ..., L(Vn)} of the vertices of C on L are ordered as L(V1), ..., L(Vn). (Property 7.1)

As will be shown in Section 7.4, this property can be used to find **all** lines on which a directed facet or an arbitrary chain can be monotonically projected. By being able to quickly find all such lines for a directed facet or an arbitrary chain, directed facets can be incrementally extended into monotone chains.

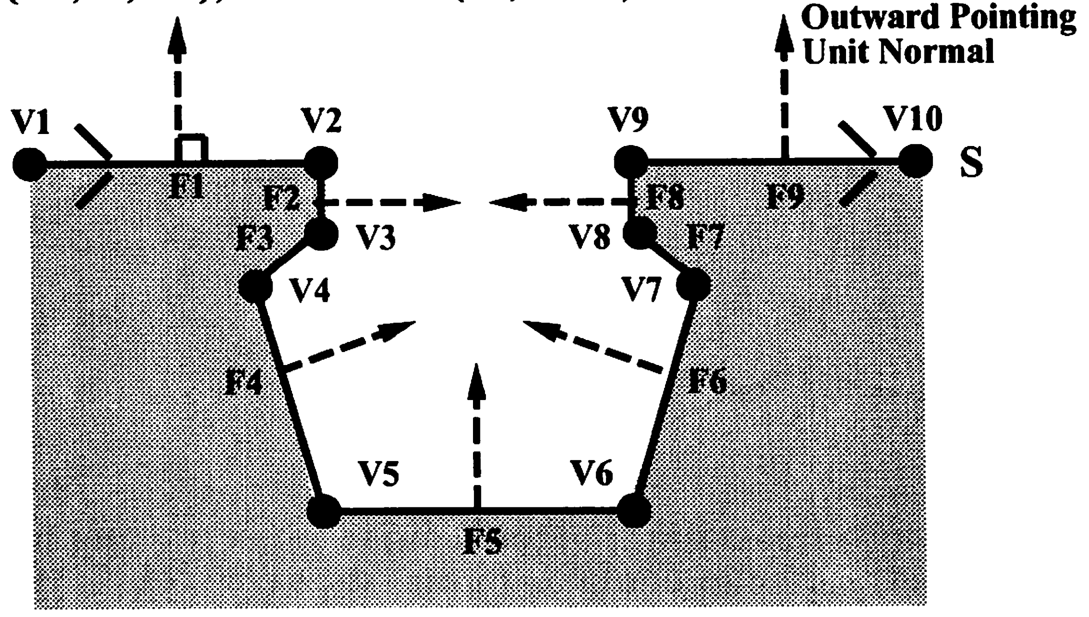
Formally, a **monotone polygon** is defined as follows [5]:

A simple polygon is said to be monotone if its boundary can be decomposed into two chains monotone with respect to the same straight line. (Definition 7.2)

This definition is illustrated in Figure 7.4, using a monotone polygon P, that consists of two monotone chains, C1 and C2. Specifically, both C1 and C2 are monotone with respect to the same line L. As Figure 7.4 suggests, surface advancement on a monotone chain usually creates another monotone chain. These two chains can be easily stitched into a monotone polygon, and used as input to boolean set operations.

Chain representing a 2D trench structure with small overhang and sloped sidewalls.

The chain C = {V1, ..., V10} has 10 directed facets {F1, ..., F9}, where Fi = (Vi, Vi+1).



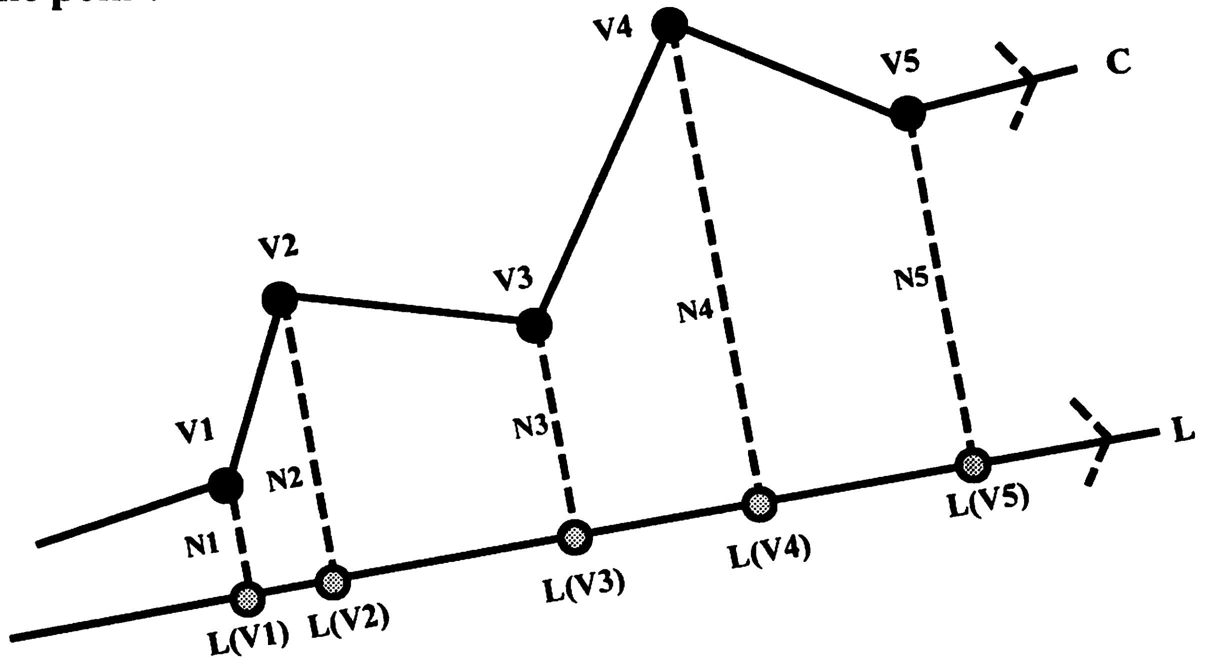
Fi is a directed facet in that when Fi is traversed as Vi, Vi+1, Fi's outward pointing normal points to its left (i.e. air lies on Fi's left and material lies on Fi's right).

Figure 7.2

Monotone Chain

[Preparata and Shamos, 1985]

A chain C = {V1, ..., Vn} is said to be monotone with respect to a straight line L if a line orthogonal to L intersects C in exactly one point.



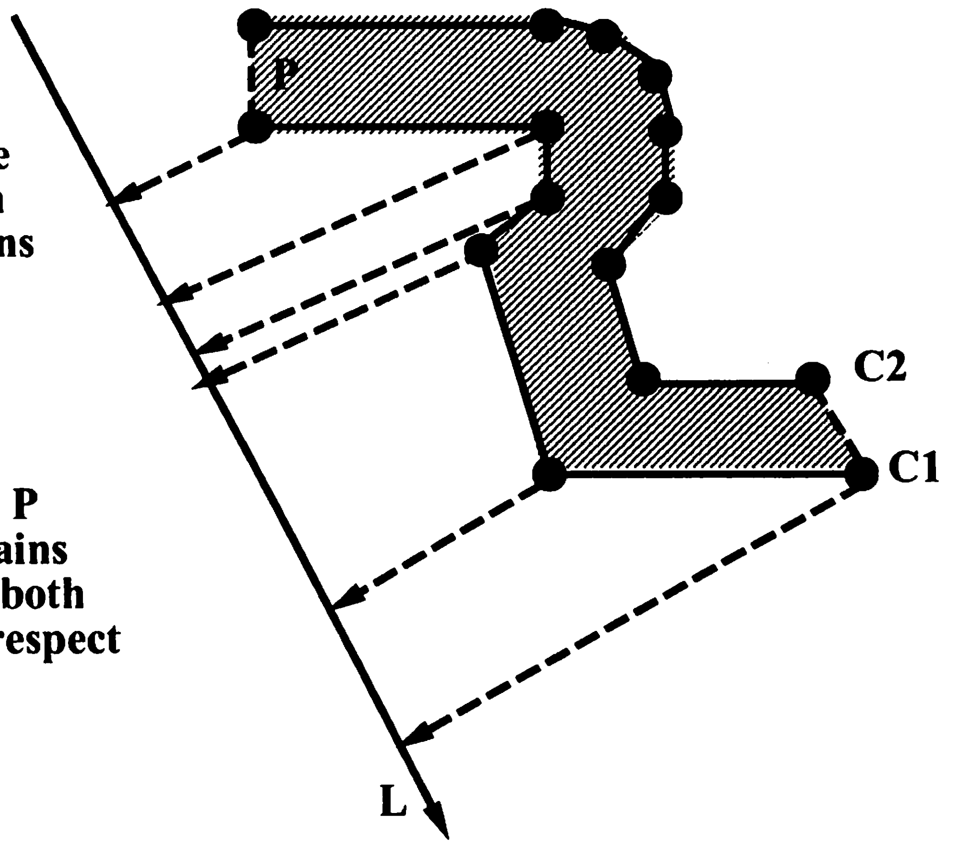
Orthogonal projections {L(V1), ..., L(Vn)} of the vertices of C on L are ordered as {L(V1), ..., L(Vn)}.

Figure 7.3

Monotone Polygon

[Preparata and Shamos, 1985]

A simple polygon is said to be monotone if its boundary can be decomposed into two chains monotone with respect to the same straight line.



Example: Monotone polygon P can be decomposed into chains C1 and C2. By inspection, both chains are monotone with respect to the straight line L.

Figure 7.4

7.4. 2D GREEDY MONOTONE DECOMPOSITION

The 2D **greedy monotone decomposition** algorithm is a heuristic that attempts to minimize the number of monotone chains. Going from left to right of the top surface, the algorithm incrementally extends monotone chains by including each surface facet in a monotone chain. The current monotone chain terminates if it reaches the right end of the top surface, or when it encounters a surface facet that cannot monotonically extend the chain. By maximally extending each monotone chain, the algorithm exploits localized facet orientation similarities inherent in IC topographies, and heuristically minimizes the number of monotone chains.

For each facet, the algorithm first computes all of its **monotone lines**, or all lines on which the facet can be monotonically projected. Computationally, directed facet Fi is monotone with respect to line L, if Fi has a positive dot product with L. This concept is illustrated in Figure 7.5 using a horizontal directed facet F1. In Figure 7.5, F1 has a positive dot product w.r.t. line L1, and a negative dot product w.r.t. line L1. Correspondingly, Figure 7.5 shows F1's vertices to be monotonically projected on line L1, but not on line L2. For computing greedy monotone decompositions, all monotone lines of a directed facet Fi can be efficiently represented using two vectors. This is because the normal directions of all of Fi's monotone lines form a half unit disc section. This section is centered around Fi's normal, and delimited by the two vectors orthogonal to Fi.

After computing the monotone lines of a directed facet, the algorithm tries to include the facet in a **monotone extension**. A monotone extension is a growing sequence of directed facets {Fi, ..., Fj}, i < j, that share at least one monotone line. Figure 7.6 depicts the expansion of a monotone extension E as it absorbs directed facets F1 through F3. As shown in

Figure 7.6, all monotone lines of a monotone extension E can also be compactly represented as unit disc sections delimited by two vectors. Computationally, the current monotone extension E can be expanded, if an overlap exists between the unit disc sections representing E's monotone lines and the next facet Fi+1's monotone lines. If E is extended by Fi+1, the overlap then represents the monotone lines of the expanded E.

The 2D greedy monotone decomposition algorithm is listed in Figure 7.7. As shown in Figure 7.7, the algorithm involves visiting each directed facet Fi on the top surface (Steps 1 and 2), computing Fi's monotone lines (Step 3), and determining whether Fi extends extension E (Steps 4 through 8). In a nutshell, the algorithm involves computing N times the intersection between the two unit disc sections representing E's monotone lines and Fi's monotone lines. (Step 6). Therefore, if this intersection can be efficiently computed, the algorithm can partition an arbitrary chain in *O(N)* time.

The intersection of unit disc sections can be computed without expensive trignometric conversions by using the concept of **pseudo angles**. As described in Karasick's thesis [12], the pseudo angle of a facet normal (N_x, N_z) is:

If $(N_z < 0)$ then PseudoAngle = $3 + N_x$; else PseudoAngle = $1 - N_x$; (7.1)

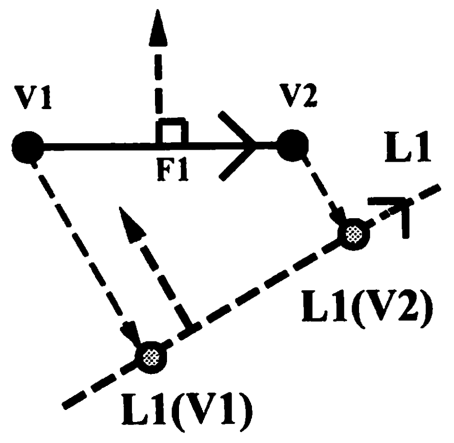
The pseudo angle formula continuously maps facet orientations from 0 to 360 degrees, into scalar values from 0 to 4. Figure 7.8 illustrates the pseudo angles for some selected facet normals. To compute the intersection between two unit disc sections, the delimiting vectors of the sections are first converted to four pseudo angles. Pseudo angles of opposing sections are pairwise compared to check for section overlaps.

An undesirable feature of greedy monotone decomposition is that it tends to decompose symmetric IC topography features asymmetrically. As will be discussed in Sections 7.7 and 7.8, monotone patch symmetry may be used to simplify the implementation of special-purpose 3D geometric computations for boundary deformation and surface visibility. As an example of this anomaly, Figure 7.9 illustrates the greedy monotone decomposition of the trench surface previously shown in Figure 7.2. In going from left to right, the algorithm inadverdently excludes the trench's local *z*-axis as a monotone line. As a result, the first monotone extension terminates at the lower right corner of the trench.

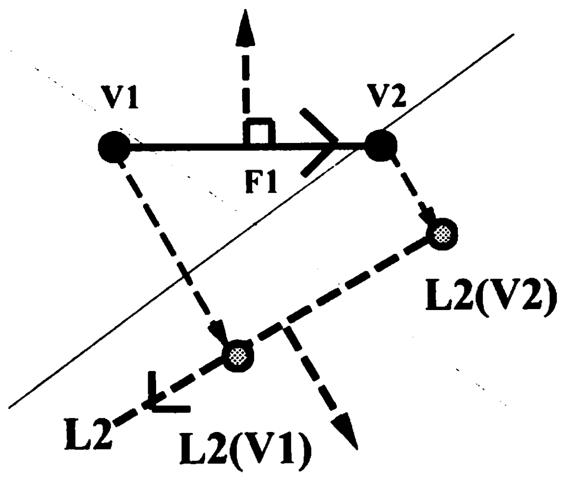
Monotone Line of a 2D Directed Facet

Computationally, line L is a monotone line of directed face Fi if Fi has a positive dot product with L.

Monotone Line L1



L2 not a Monotone Line



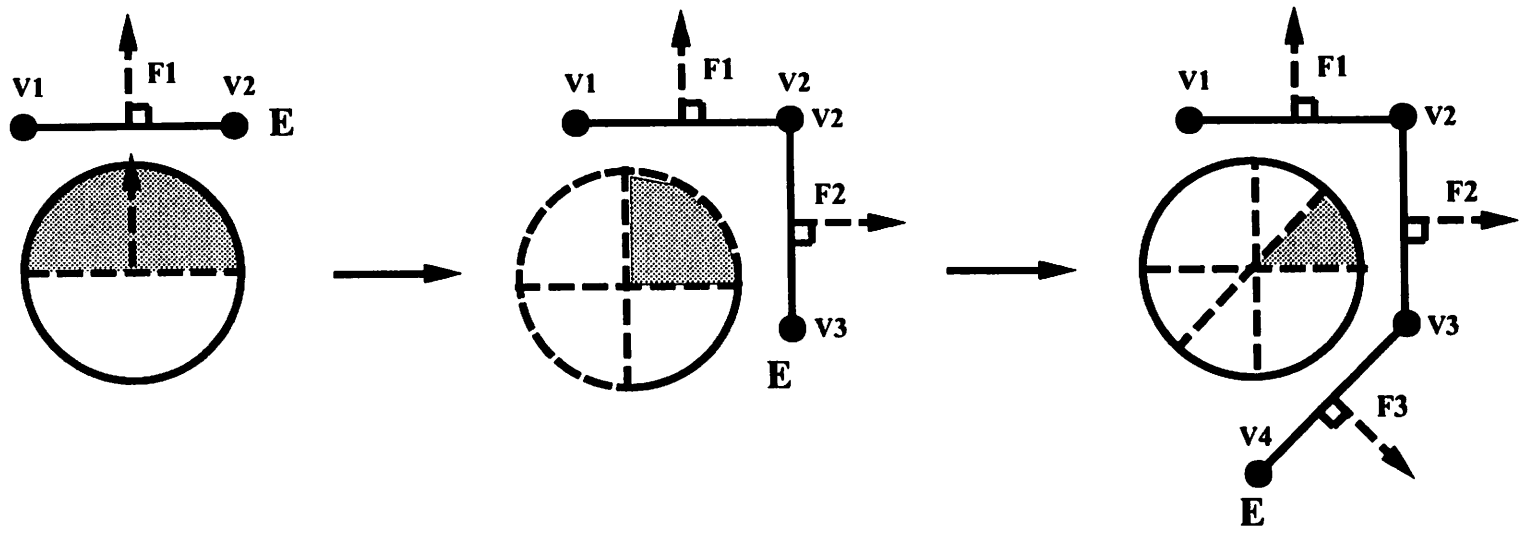
Orthogonal projections of Vi, Vi+1 on monotone line L can be traversed as L(Vi), L(Vi+1).

Figure 7.5

Figure 7.6

Monotone Extensions of 2D Directed Facets

Monotone extension E is a growing sequence of directed facets {Fi, ..., Fj}, where i < j, which share at least one monotone line.



E's unit disc section becomes more constricted as facets with different orientations are included.

2D Greedy Monotone Decomposition

```
1. Initialize E, the current monotone extension,
   to nil. Initialize De, the unit disc section repre–
   senting E's monotone lines, to full unit disc.

2. For each directed facet Fi,

3.   Compute Dfi, the unit disc section
     representing Fi's monotone lines.

4.   If E is nil,
       Include Fi as the initial facet of E.
       Set De to Dfi.
       Go to Step 2.

5.   Else /* Monotone extension in progress. */

6.     Set De_test = De intersect Dfi.

7.     If De_test is nil /* Can't monotone extend. */
         Report E as a monotone string.
         Re–initialize E and De (as in Step 1).
         Include Fi as the initial facet of E.
         Set De to Dfi.
         Go to Step 2.

8.     Else /* Monotone extend E */
         Include Fi as next facet of E.
         Set De to De_test.
         Go to Step 2.
```

Figure 7.7

RHW – UCB TCAD

Pseudo Angles for Computing Unit Disc Section Intersections

If (z < 0) then (Angle = 3 + x) else (Angle = 1 − x).

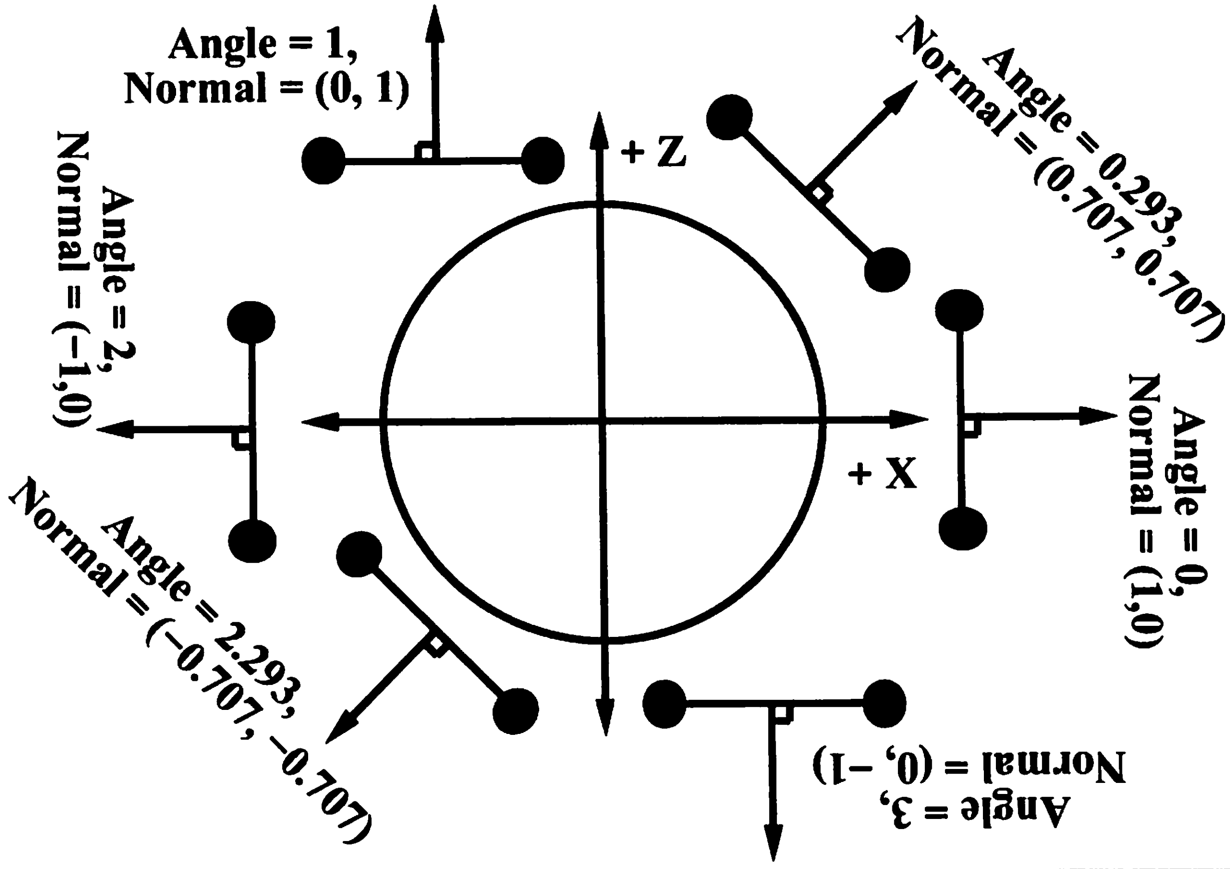
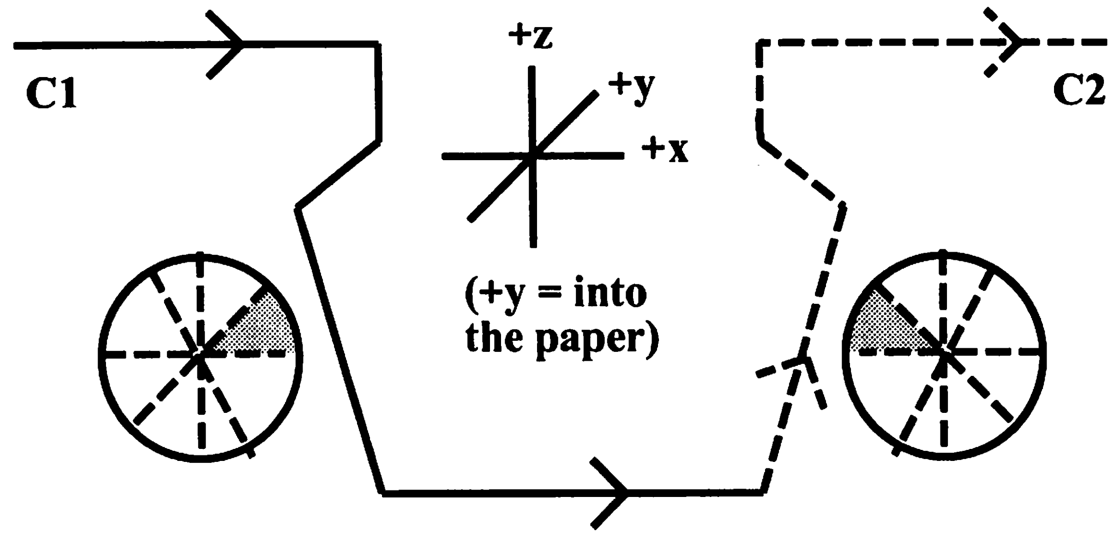


Figure 7.8

Figure 7.9

Greedy Monotone Decomposition may fail to include a feature's local z–axis as a monotone line, thereby asymmetrically decompose the feature.



7.5. THE 2.5D ISOTROPIC DEPOSITION EXPERIMENT

The **2.5D Isotropic Deposition Experiment** was designed to evaluate 2D greedy monotone decomposition. In the previous section, it was shown that greedy monotone decomposition heuristically attempts to minimize the number of monotone chains. This in turn allows aggregate geometrical operations, such as boundary deformation, to invoke a minimal number of CPU intensive primitive geometrical operations, such as boolean set operations. On the other hand, in going from left to right of the top surface, the algorithm may asymmetrically partition symmetric IC topographical features, such as trenches. As a demonstration of these algorithmic properties, the **2.5D Isotropic Deposition Experiment** uses IBM Geometry Engine (3D) boolean set operations to resolve global intra-surface collisions inherent in the use of 2D surface-based IC topography simulators.

In the **2.5D Isotropic Deposition Experiment**, Hamaguchi's 2D shock tracking solver [7] was used to simulate a 0.3 um isotropic deposition on a 1 um deep 2D key hole trench with a 0.25 um opening at the top of the trench. Figure 7.10 plots snapshots of the simulation result at time (T) = 0, 1, 2, and 3 seconds. (Note that the scales of the *x*-axis and the *z*-axis are not 1:1.) As shown in Figure 7.10, at T = 1 second, the isotropic deposition created a void in the topography. This example showed that the shock tracking simulator could numerically detect and remove local loops near concave corners at the bottom of the trench. In this sense, it is more sophisticated than conventional surface-based IC topography simulators, such as SAMPLE [8][9], SPEEDIE [10], and several other corner and facet pushing programs [11]-[14]. However, the simulator still could not detect global loops at the top of the trench. More importantly, the failure to remove global loops led to inaccurate simulation of the size and shape of the resulting void.

In the **2.5D Isotropic Deposition Experiment**, 2D greedy monotone decomposition was used to minimize the number of IBM Geometry Engine **merge** operations used to resolve global collisions. Figure 7.11 illustrates how monotone decomposition can be used to sweep a surface along its shock traces, and into a few monotone polygons. First, as shown on the left of Figure 7.11, shock traces are computed on the surface. Then, as depicted in the middle of Figure 7.11, the surface is partitioned into two monotone chains, and the chains are swept along shock traces into two monotone polygons. Finally, as shown on the right of Figure 7.11, the monotone polygons are extruded into volumes and merged by the IBM Geometry Engine into an aggregate deformation volume. As discussed at the end of Section 7.4, greedy monotone decomposition asymmetrically partitioned the 2D key hole trench at the lower right corner of the trench.

To demonstrate the performance improvement gained from using monotone decomposition, the trench surface was separately swept into triangles and qudrilaterals, and into monotone polygons. The number of operations and total CPU times used by the IBM Geometry Engine to merge these polygons were compared. Figure 7.12 depicts the polygons swept from the trench surface, and the corresponding CPU times used by IBM Geometry Engine **merge** operations running on an IBM RS/6000 Model 530 workstation with a 32 MB RAM. Without monotone decomposition, as shown on the left of Figure 7.12, 48 operations and a total of 22.42 seconds were used to merge triangular and quadrilateral prisms. With monotone decomposition, as shown on the right of Figure 7.12, 2 operations and a total of 1.04 seconds were used to merge monotone polygonal prisms. Although the average CPU time was slightly higher for merging large grain monotone polygonal prisms (0.52 seconds versus 0.47 seconds), the reduction in number of operations (24x) led to a similar magnitude reduction in total CPU time (22x).

Isotropic Deposition on 2D Key Hole Trench – Shock Trace Advancement

* Local loops may be removed numerically, but not global loops.

* Global loops should be resolved immediately to simulate accurately void size and shape.

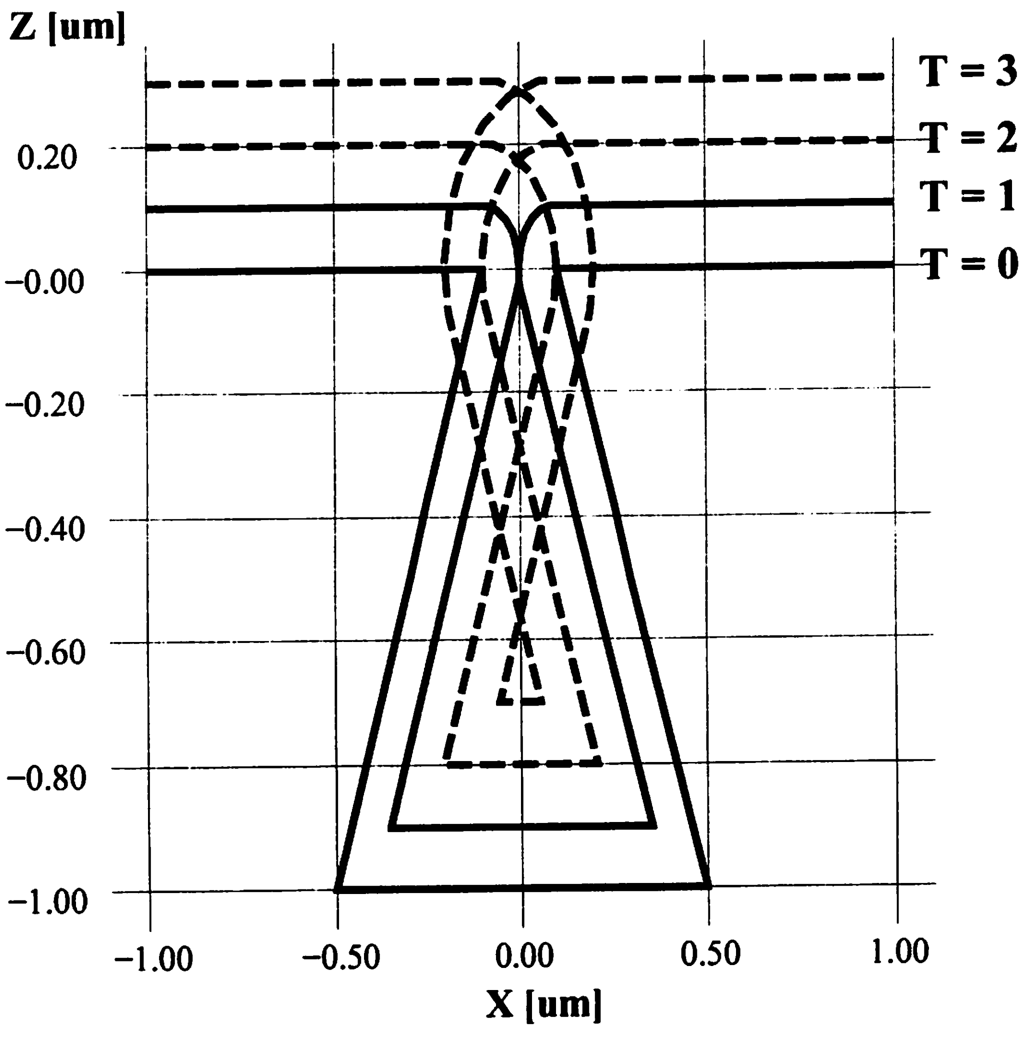


Figure 7.10

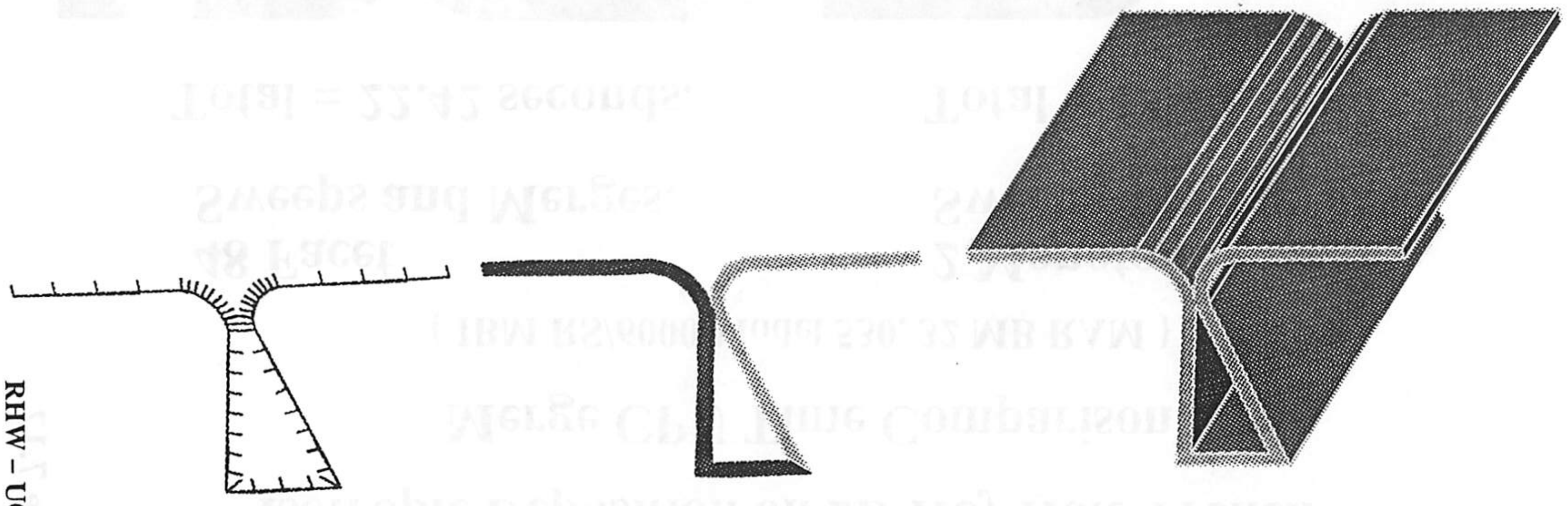
RHW – UCB TCAD

Figure 7.11

Isotropic Deposition on 2D Key Hole Trench

Simulation Objects

Shock–Traces **Monotone Polygons** **Deformation Volume**



RHW – UCB TCAD

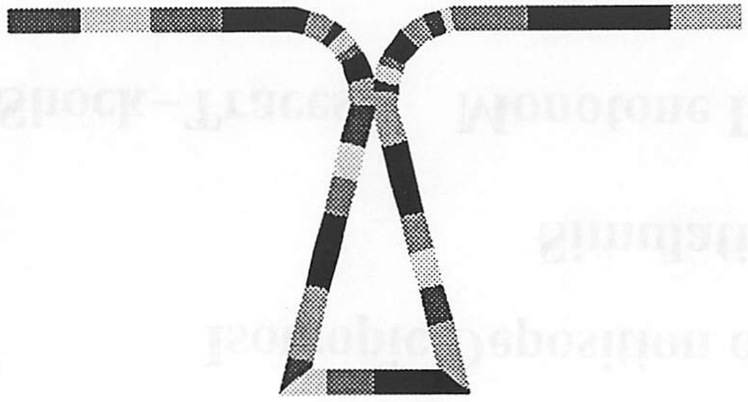
Isotropic Deposition on 2D Key Hole Trench

Merge CPU Time Comparison

{ IBM RS/6000 Model 530, 32 MB RAM }

48 Facet
Sweeps and Merges.

Total = 22.42 seconds.



2 Monotone Chain
Sweeps and Merges.

Total = 1.04 seconds.

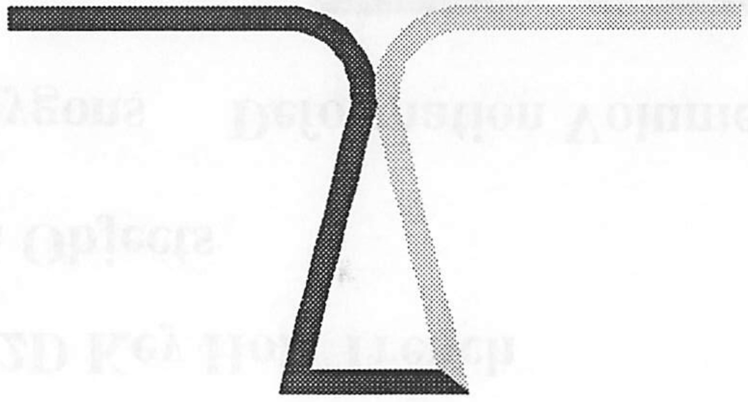


Figure 7.12

7.6. 3D DIRECTED MONOTONE DECOMPOSITION

By predefining monotone planes, the 3D **directed monotone decomposition** algorithm can axially symmetrically decompose IC topographical features. The algorithm implicitly chooses monotone planes that partition facet orientations symmetrically about the local *z*-axes of topographical features. Using facet orientation partitions, the algorithm first classifies each facet as **left-pointing**, **right-pointing**, or **horizontal**, as well as **upward-pointing**, **downward-pointing**, or **vertical**. A breadth first traversal is then used to group locally connected facets that have the same orientation classifications. Like the 2D greedy monotone decomposition algorithm, the 3D directed monotone decomposition algorithm also exploits localized facet orientation similarities in IC topographies. By using breadth first traversal to group facets, the algorithm implicitly partitions the simulated surface by topographical features as well as facet orientations.

The 3D directed monotone decomposition algorithm currently creates facet orientation partitions along the local *yz*- and *xy*-planes of IC topographical features. Figure 7.13 plots the facet orientation partitions used by the algorithm. In Figure 7.13, a facet normal is denoted by (N_x, N_y, N_z). As shown in Figure 7.13, the *yz*-plane partitions facet orientations into five bins as **left-pointing** $(N_x < 0)$, **front-pointing)** $(N_x == 0$ and $N_y < 0)$, **right-pointing** $(N_x > 0)$, **(back-pointing)** $(N_x == 0$ and $N_y > 0)$, and **horizontal** (not shown) $\{(N_x == 0$ and $N_y == 0)$ and $(N_z \mathrel{!=} 0)\}$. The *yz*-plane further partitions facet orientations into three sub-bins as **upward-pointing** $(N_z > 0)$, **downward-pointing** $(N_z < 0)$, and **vertical** (not shown) $\{(N_z == 0)$ and $(N_x \mathrel{!=} 0$ or $N_y \mathrel{!=} 0)\}$. This main advantage of this straightforward partitioning scheme is that facets can be efficiently classified by examining the signs of facet normal components.

Using these facet orientation bins, two directed monotone decomposition algorithms have been implemented: *X*-cut monotone decomposition, and *Xz*-cut monotone decomposition. The **x-cut monotone decomposition** algorithm looks mainly at the *x*-component (N_x) of the facet normal. The algorithm classifies facets using one of **three** categories: **1) Left-pointing or front-pointing, 2) Right-pointing or back-pointing,** and **3) Horizontal.** Figure 7.14 illustrates a x-cut monotone decomposition of a spherical trench surface with 380 triangles. The left of Figure 7.14 shows the IBM Geometry Engine solid model used to extract the spherical trench surface. On the right of Figure 7.14, the spherical trench surface is decomposed into 6 monotone patches using x-cut monotone decomposition. As shown in Figure 7.14, spherical trench surfaces are excellent examples for testing directed monotone decomposition because the surface is symmetric about its local *z*-axis, and contains a wide range of facet orientations.

The **xz-cut monotone decomposition** algorithm looks at both the *x*-component (N_x) and *z*-component (N_z) of the facet normal. The algorithm classifies facets using one of **eight** categories: **1a) Left-and-upward-pointing, 1b) Left-and-downward-pointing, 1c) Vertical-and-Left-pointing, 2a) Right-and-upward-pointing, 2b) Right-and-downward-pointing, 2c) Vertical-and-Right-pointing, 3a) Horizontal-and-upward-pointing,** and **3b) Horizontal-and-downward-pointing.** Figure 7.15 illustrates a xz-cut monotone decomposition of the same spherical trench surface with 380 triangles. The left of Figure 7.15 again depicts the spherical trench solid model. On the right of Figure 7.15, the spherical trench surface is decomposed into 10 monotone patches using xz-cut monotone decomposition. As expected, increasing the number of facet orientation groups resulted in a few more monotone patches.

The 3D directed monotone decomposition algorithm is listed in Figure 7.16. As shown in Figure 7.16, the algorithm decomposes an arbitrary surface using two traversals of all surface facets. Each surface traversal was designed to support any facet orientation partitioning scheme. The first traversal (Steps 1 and 2), walks through the facet list, and classifies each facet according to its orientation. Steps 3 through 6 then perform the breadth first traversal that groups similiarly oriented triangles into monotone patches. Since the algorithm involves only floating point and integer comparisons, it runs efficiently in *O(N)* time.

Despite using coarse facet orientation partitions, directed monotone decomposition can decompose complex topographies by topographical features as well as facet orientations. Figure 7.17 illustrates a x-cut monotone decomposition of a periodic spherical trench surface with 1,488 triangles. The left of Figure 7.17 shows the IBM Geometry Engine solid model used to extract the periodic spherical trench surface. On the right of Figure 7.17, the periodic spherical trench surface is decomposed into 8 monotone patches using x-cut monotone decomposition. As expected, the directed monotone decomposition algorithm can axially symmetrically partition individual trenches in the periodic structure.

Facet Orientation Classifications for 3D Directed Monotone Decomposition

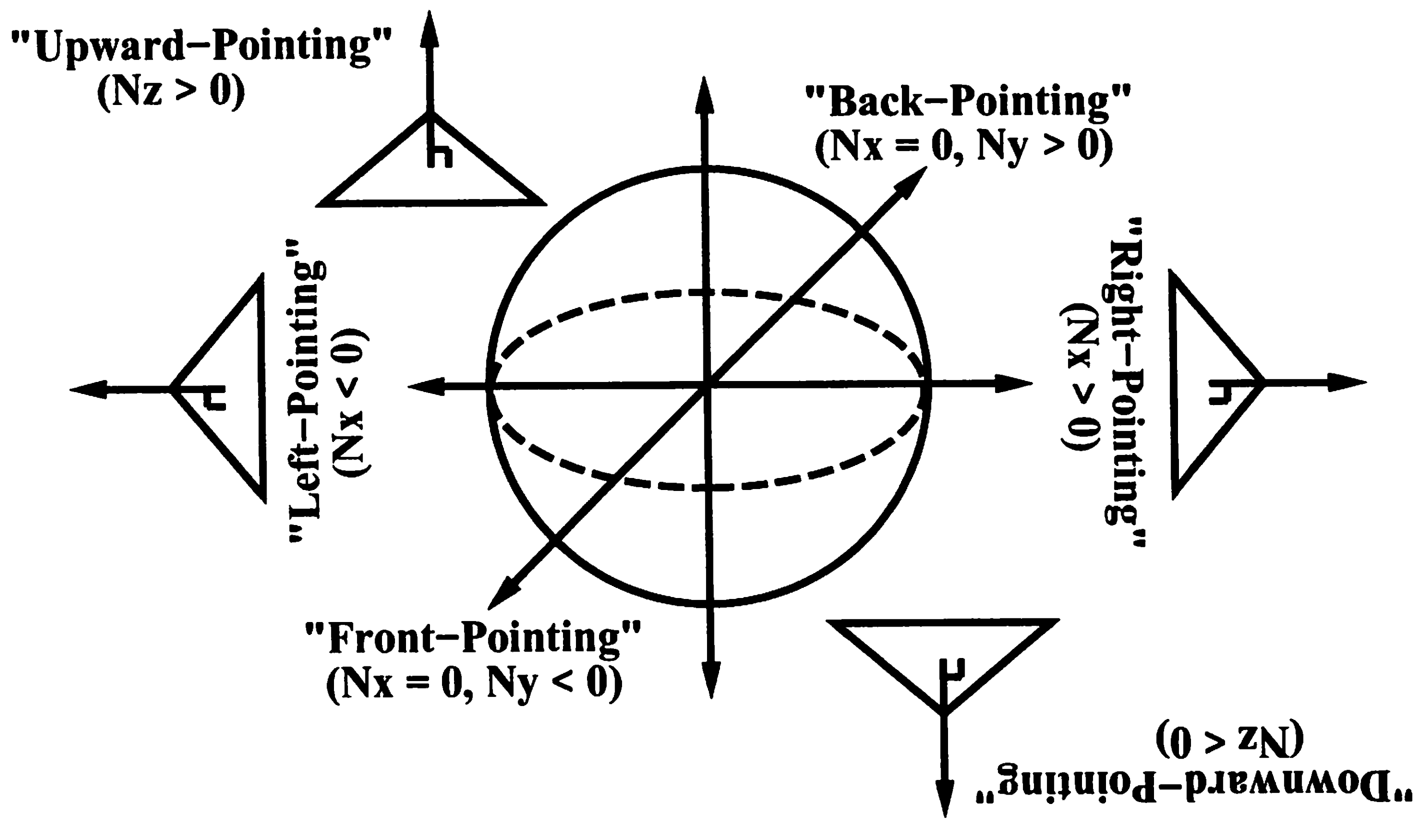
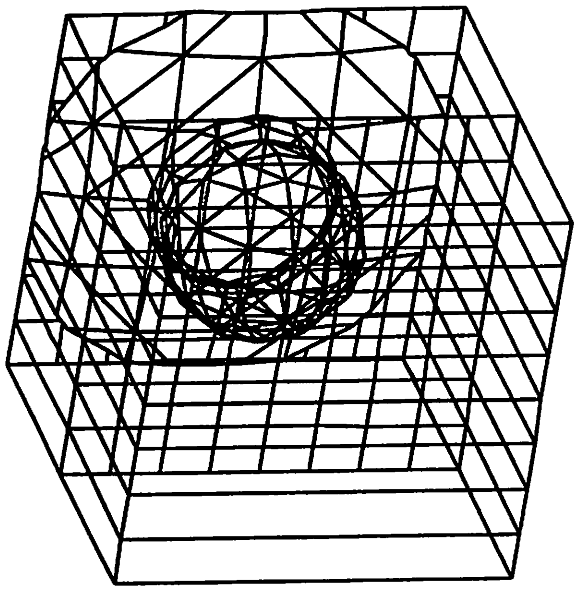


Figure 7.13

"X Cut" Monotone Decomposition of Spherical Trench



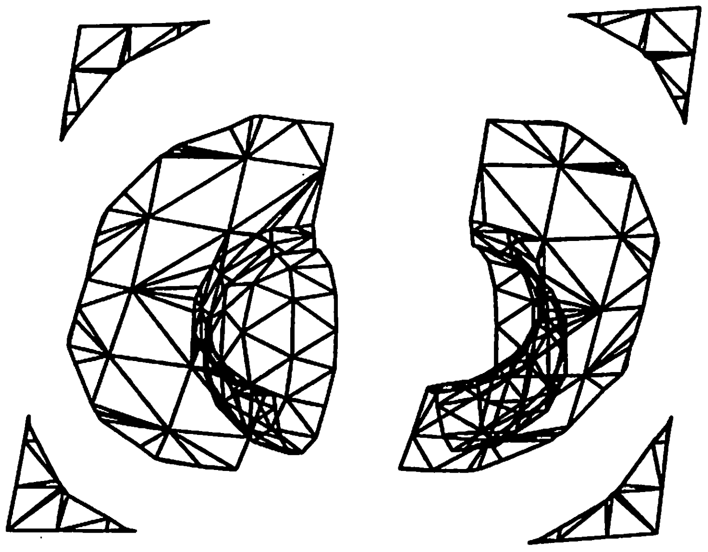
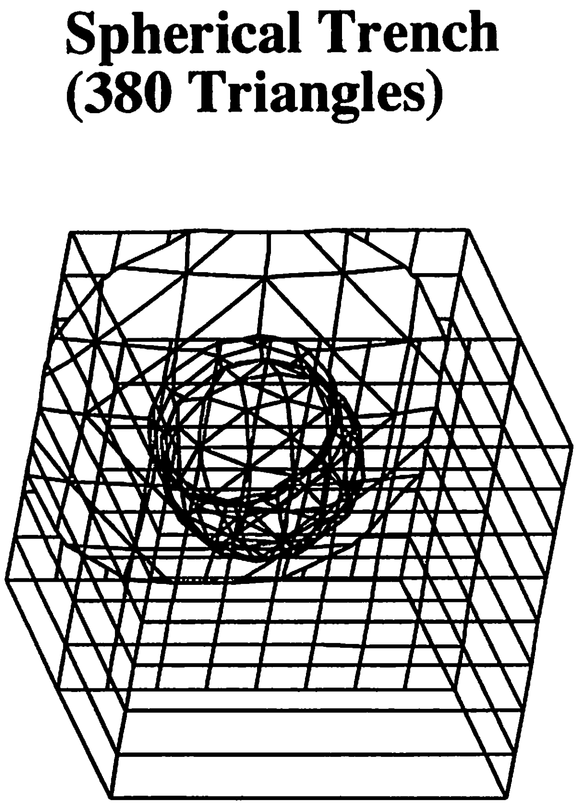


Figure 7.14

"XZ Cut" Monotone Decomposition of Spherical Trench



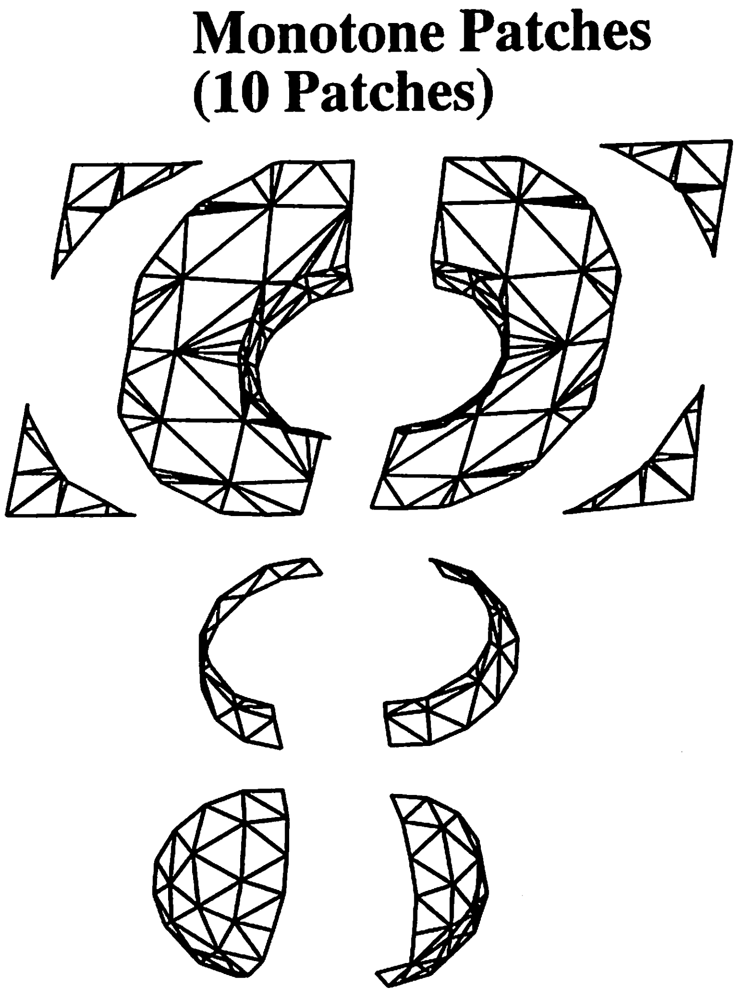


Figure 7.15

3D Directed Monotone Decomposition

1. For each surface mesh triangle T with outward pointing normal (Nx, Ny, Nz),

2a. ("X Cut" Decomposition)
Classify T as Left_or_Front Pointing, Right_or_Back Pointing, and Horizontal by looking mainly at sign of Nx's.

2b. ("XZ Cut" Decomposition)
Add also Up Pointing, Down Pointing, and Vertical sub-cases to each case in "X Cut" Decomposition as appropriate.

3. For each surface mesh triangle T with traversal marker,

4. If T is not traversed,

5. Starting at T, recursively traverse and mark unmarked triangle neighbors with the same classification.

6. Create monotone patch from traversed triangles.

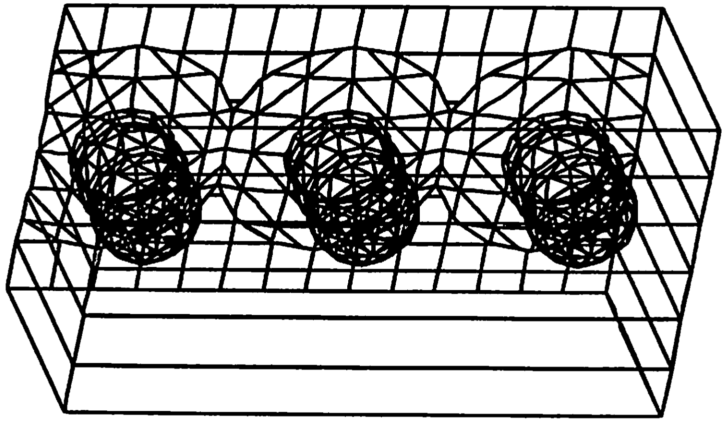
Figure 7.16

RHW - UCB TCAD

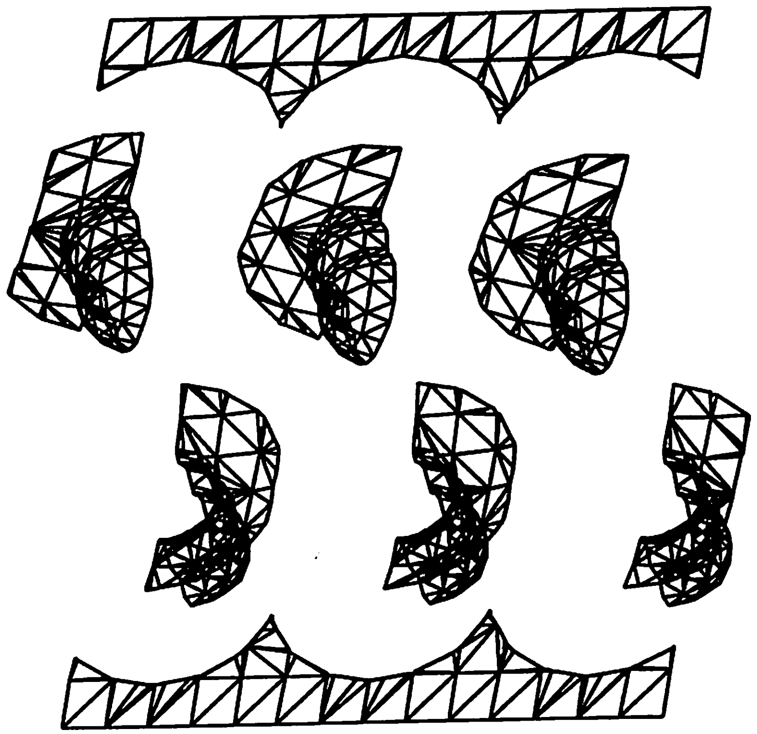
Figure 7.17

By predefining monotone planes, Directed Monotone Decomposition axial symmetrically partitions features.

Periodic Spherical Trench
(1,488 Triangles)



Monotone Patches
(8 Patches)



RHW – UCB TCAD

7.7. THE 3D ISOTROPIC DEPOSITION EXPERIMENT

The **3D Isotropic Deposition Experiment** is the full 3D equivalent of the **2.5D Isotropic Deposition Experiment**. The test was designed to show that directed monotone decomposition makes it feasible to use IBM Geometry Engine **merge** operations in 3D boundary deformation. In 2D, since surfaces contain relatively small numbers (about 100) of facets, monotone decomposition was helpful but not essential in enabling the use of the IBM Geometry Engine. In 3D, a simulated surface typically contains 1,000 to 10,000 facets. As shown by the **Localized Deformation Test** in Section 6.6, it is impractical to invoke such large numbers of IBM Geometry Engine **merge** operations to simulate boundary deformation at each time step. Therefore, for 3D boundary deformation, large grain surface decomposition methods, such as monotone decomposition, are essential to the use of the IBM Geometry Engine. Furthermore, to simplify the extraction of monotone deformation volumes, directed monotone decomposition is needed to partition surfaces axially symmetrically.

The **3D Isotropic Deposition Experiment** demonstates the need for directed monotone decomposition using SAMPLE-3D as a 3D surface-based IC topography simulator. In this test, SAMPLE-3D was used to simulate a 0.6 um isotropic deposition on a 2 um deep 3D key hole trench with a 1 um opening at the top of the trench. Simulations were carried out on trench surfaces with 452 and 896 triangles. Figure 7.18 plots the initial and final surfaces at T = 6 seconds for 896 triangles. As shown in Figure 7.18, the deposition created a void that was shaped like an inverted cone. In this simulation, loops occurred at the bottom as well as at the top of the trench. As depicted in Figure 7.18, the extraneous loops at the bottom of the trench were caused by facet motions from the trench sidewalls and the trench bottom, and formed an inverted donut.

In the **3D Isotropic Deposition Experiment**, directed monotone decomposition was used to reduce the number of IBM Geometry Engine **merge** operations used to resolve global collisions. Figure 7.19 illustrates how monotone decomposition was used to sweep the simulated surface along its vertex deformation vectors, and into a few monotone volumes. The left of Figure 7.19 shows the IBM Geometry Engine solid model used to extract the 3D key hole trench surface. On the right of Figure 7.19, the key hole trench surface is decomposed into 9 monotone patches, and these patches were swept into 9 new patches along vertex deformation vectors. As shown in Figure 7.19, a sophisticated solid extraction operation is needed here to preserve the void and eliminate extraneous loops. Geometric utilities that perform this functionality are currently being developed for capacitance extraction from SAMPLE-3D simulated surfaces [15].

To demonstrate the performance improvement that could be gained from using directed monotone decomposition, average CPU times for merging a triangular prism or a monotone deformation volume were calculated by dividing the CPU times obtained from the **Localized Deformation Test** by their corresponding number of **merge** operations. Figure 7.20 plots the average CPU times per **merge** operation for constructing staircases with 100 to 1600 cubes. In Figure 7.20, average CPU times were calculated for input data granularities of 1 cube and $\sqrt{N}$ cubes. As shown in Figure 7.20, for 450 cubes, each **merge** operation takes about 3 seconds for both granularities. For 900 cubes, each **merge** operation takes about 6 seconds for both granularities.

Using these average CPU times, the use of the IBM Geometry Engine to merge triangular prisms versus monotone deformation volumes were compared in terms of numbers of **merge** operations and estimated total CPU times. Tables 7.1ab summarize this performance comparison. Table 7.1a lists the reduction in numbers of **merge** operations, and the increase in memory consumption. Table 7.1b lists the corresponding reduction in total CPU times used by IBM Geometry Engine **merge** operations, and CPU times used to monotonicially decompose surfaces. For a 3D key hole trench with 896 triangles, the number of **merge** operations was reduced from 895 down to 8. Correspondingly, total CPU time would be reduced from about 5,370 seconds to about 48 seconds. Therefore, with monotone decomposition, the IBM Geometry Engine could efficiently merge a 1,000 facet aggregate deformation volume in about 1 minute of CPU time.

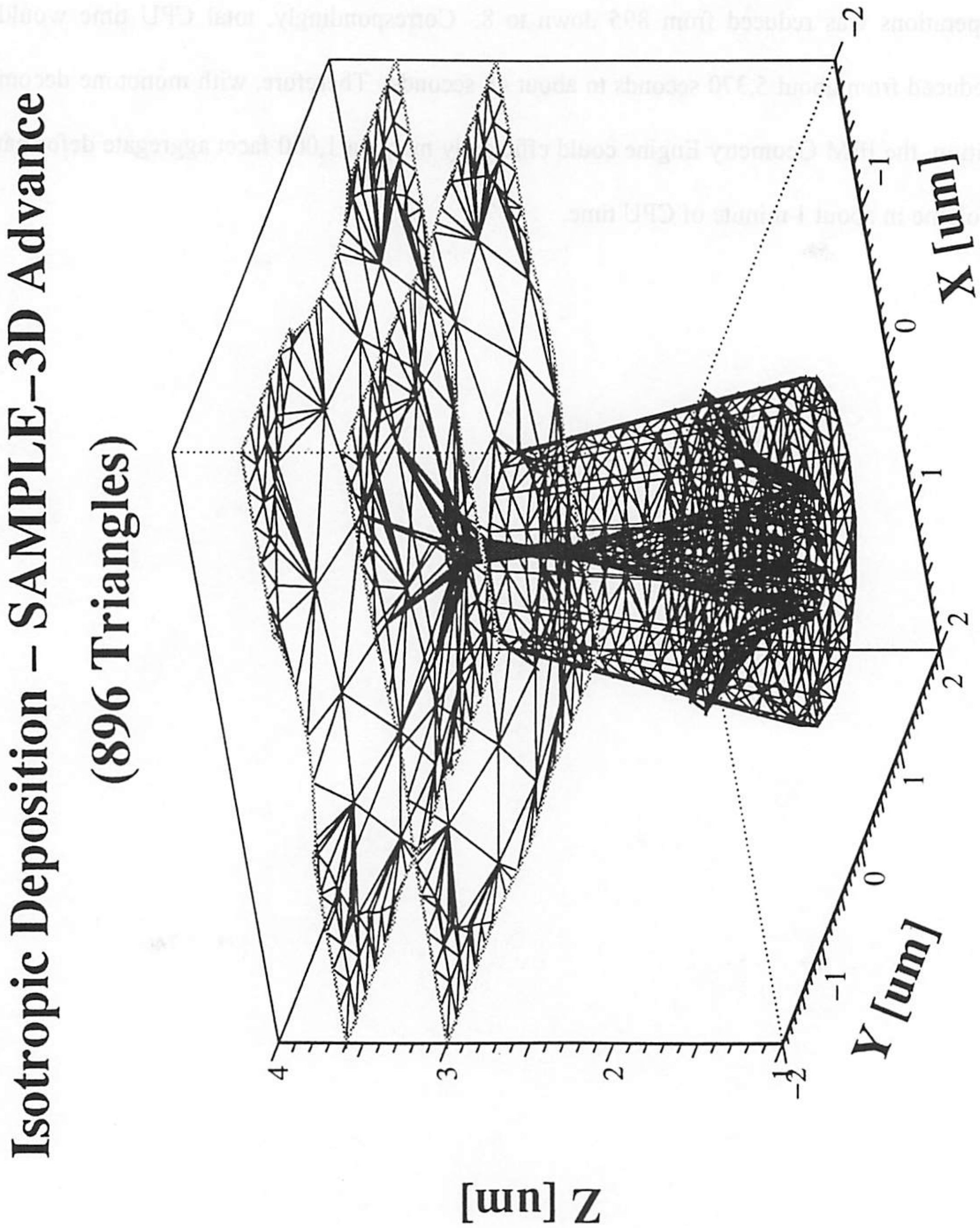
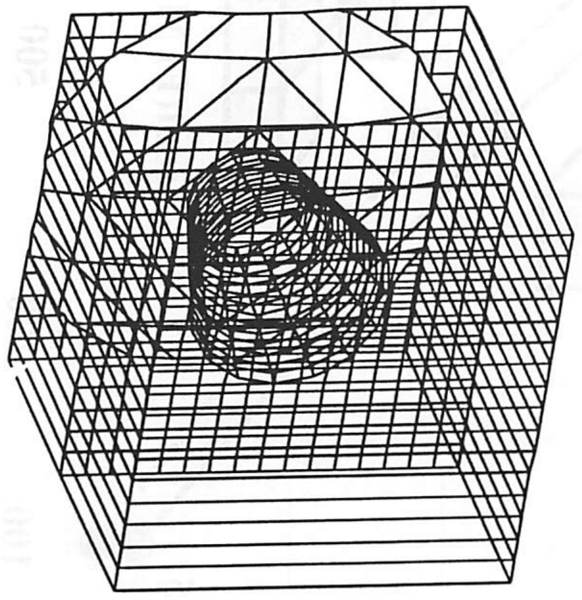


Figure 7.18

RHW – UCB TCAD

Isotropic Deposition – Monotone Decomposition
(896 Triangles)

Refined 3D Key Hole Trench



Initial + Advanced Monotone Surfaces

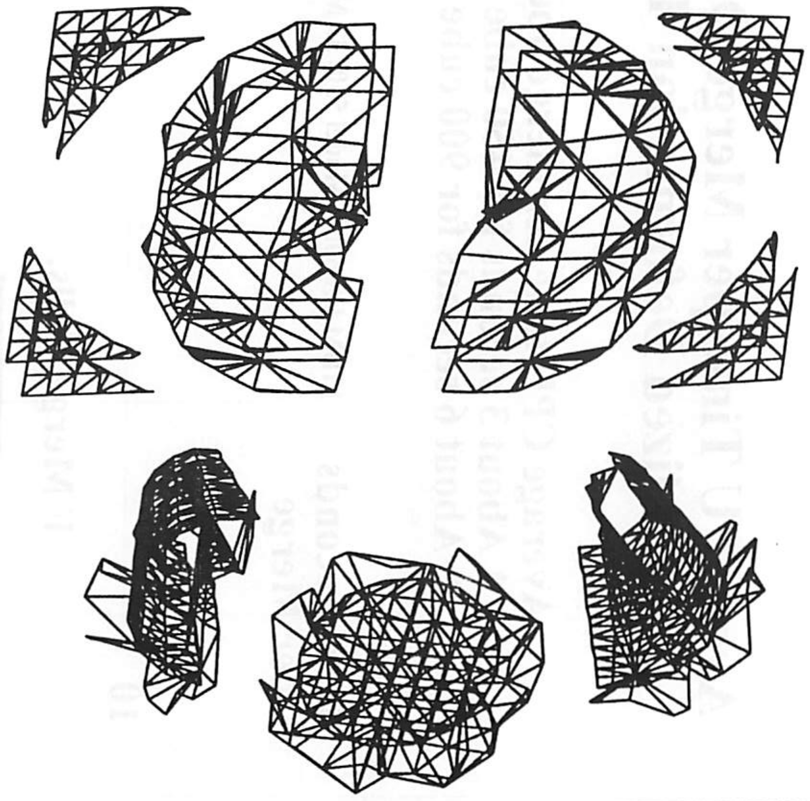


Figure 7.19

RHW – UCB TCAD

Avg CPU Time per Merge Operation: Localized Deformation Test

Average CPU time per merge operation:
* About 3 seconds for 450 cube case.
* About 6 seconds for 900 cube case.

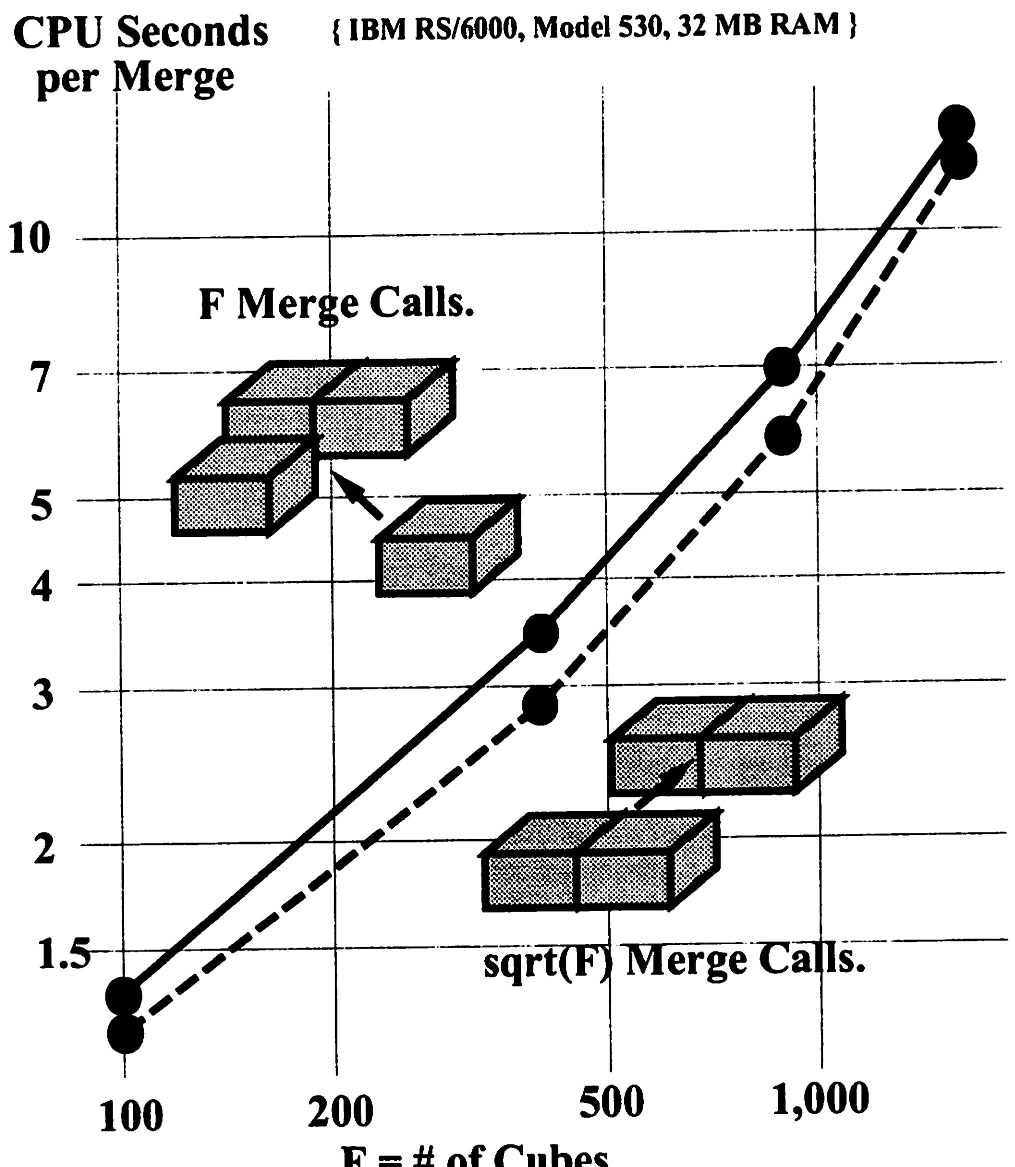


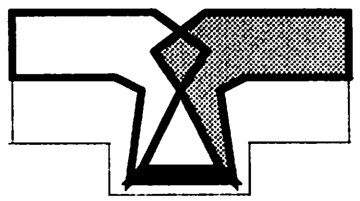
Figure 7.20

RHW - UCB TCAD

Table 7.1a

Isotropic Deposition on Key Hole Trench with and w/o Monotone Decomposition

{ IBM RS/6000, Model 530, 32 MB RAM }



# of Merge Calls Comparison

Number of Triangles	With Monotone	No Monotone
452	8	451
896	8	895

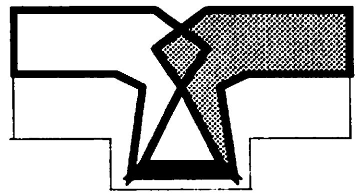
Memory Consumption Comparison

Number of Triangles	With Monotone	No Monotone
452	3,352 kB	2,476 kB
896	6,192 kB	4,696 kB

Table 7.1b

Isotropic Deposition on Key Hole Trench with and w/o Monotone Decomposition

{ IBM RS/6000, Model 530, 32 MB RAM }



CPU Time Comparison

Number of Triangles	Monotone Decomp.	With Monotone (Estimated)	No Monotone (Estimated)
452	6.03 sec	24 sec	1,353 sec
896	17.29 sec	48 sec	5,370 sec

7.8. THE 3D SOURCE VISIBILITY EXPERIMENT

The **3D Source Visibility Experiment** was designed to demonstrate how directed monotone decomposition can improve the efficiency in using SAMPLE-3D **line-of-sight visibility tests** to compute surface visibility. In a typical 3D IC topography simulation, a surface mesh contains about 1,000 to 10,000 vertices, and a hemispherical source is represented by about 100 to 500 points. If source visibility is rigorously calculated using **line-of-sight visibility tests** at every surface vertex, a typical simulation may use millions of tests. Directed monotone decomposition can increase the data granularity of surface visibility computation from surface vertices to monotone patches. For each monotone patch, **line-of-sight visibility tests** can be first applied at a few representative monotone patch vertices. These rigorously computed source visibilities are then interpolated over each monotone patch.

The **3D Source Visibility Experiment** explored the use of directed monotone decomposition to improve the efficiency of using **line-of-sight visibility tests** in SAMPLE-3D source visibility computation. Using a hemispherical source with 45x10 points, SAMPLE-3D first computed the source visibility of spherical trench surfaces with 380 and 904 triangles without using monotone decomopsition. In other words, source visibility was rigorously computed using **line-of-sight visibility tests** at every surface vertex. Figure 7.21 plots the qualitative results for 380 triangles. In Figure 7.21, darker triangle shading indicates more limited source visibility. As expected, source visibility is most restricted at the bottom of the spherical trench surface.

In the **3D Source Visibility Experiment**, directed monotone decomposition was used to reduce the number of **line-of-sight visibility tests** in SAMPLE-3D source visibility computation. For each monotone patch, the test involved: 1) Selecting a few representative monotone patch vertices, 2) Rigorously computing the source visibilities at the selected vertices by applying SAMPLE-3D **line-of-sight visibility tests**, and 3) Interpolating the source visibilities over the monotone patch. In this test, monotone patch vertices nearest to the center and bounding box corners of the monotone patch were selected for rigorous source visibility calculation. Since the goal of this test was to characterize the performance improvement gained from reducing the number of SAMPLE-3D **line-of-sight visibility tests**, monotone patch source visibility was assumed to be constant, and was equal to the average of the rigorously computed source visibilities.

Figure 7.22 plots the source visibility of the 380-triangle spherical trench surface computed using monotone decompostion. From this plot, it is clear that the constant interpolation of rigorously computed source visibilities significantly over-estimated the source visibility at the bottom of the trench. To make better use of monotone decomposition, special-purpose algorithms are needed to accurately and efficiently interpolate monotone patch source visibility. Development of these algorithms would involve customizing shading interpolation algorithms from computer graphics, such as the ones described in [16], for the special case of symmetric monotone patches. To efficiently interpolate monotone patch source visibility, these algorithms should exploit facet orientation similarity within each patch, as well as spatial coherence between patches from the same topographical feature.

The use of SAMPLE-3D **line-of-sight visibility tests** to compute source visibility without versus with monotone decomposition were compared in terms of numbers of tests and total CPU times. Tables 2ab summarize this performance comparison. Table 7.2a lists the reduction in numbers of **line-of-sight visibility tests**, and the increase in memory consumption. Table 7.2b lists the corresponding reduction in total CPU times used by SAMPLE-3D **line-of-sight visibility tests**, and CPU times used to monotonically decompose surfaces. For a 3D spherical trench surface with 904 triangles, the number of **line-of-sight visibility tests** was reduced from 75,554 down to 8,176. Correspondingly, total CPU time was reduced from 9.30 seconds down to 1.34 seconds. Theoretically, with monotone decomposition, the number of **line-of-sight visibility** tests can reduce from *O(N*S)*, where S is the number of source points, to *O(M*v*S)*, where M is the number of monotone patches and v is the number of monotone patch vertices. Since each surface vertex can issue between 45 to 450 **line-of-sight visibility tests** to calculate its source visibility, further reduction in the number of **line-of-sight visibility tests** will depend on the ability to minimize the number of monotone patch vertices selected for rigorous source visibility calculations.

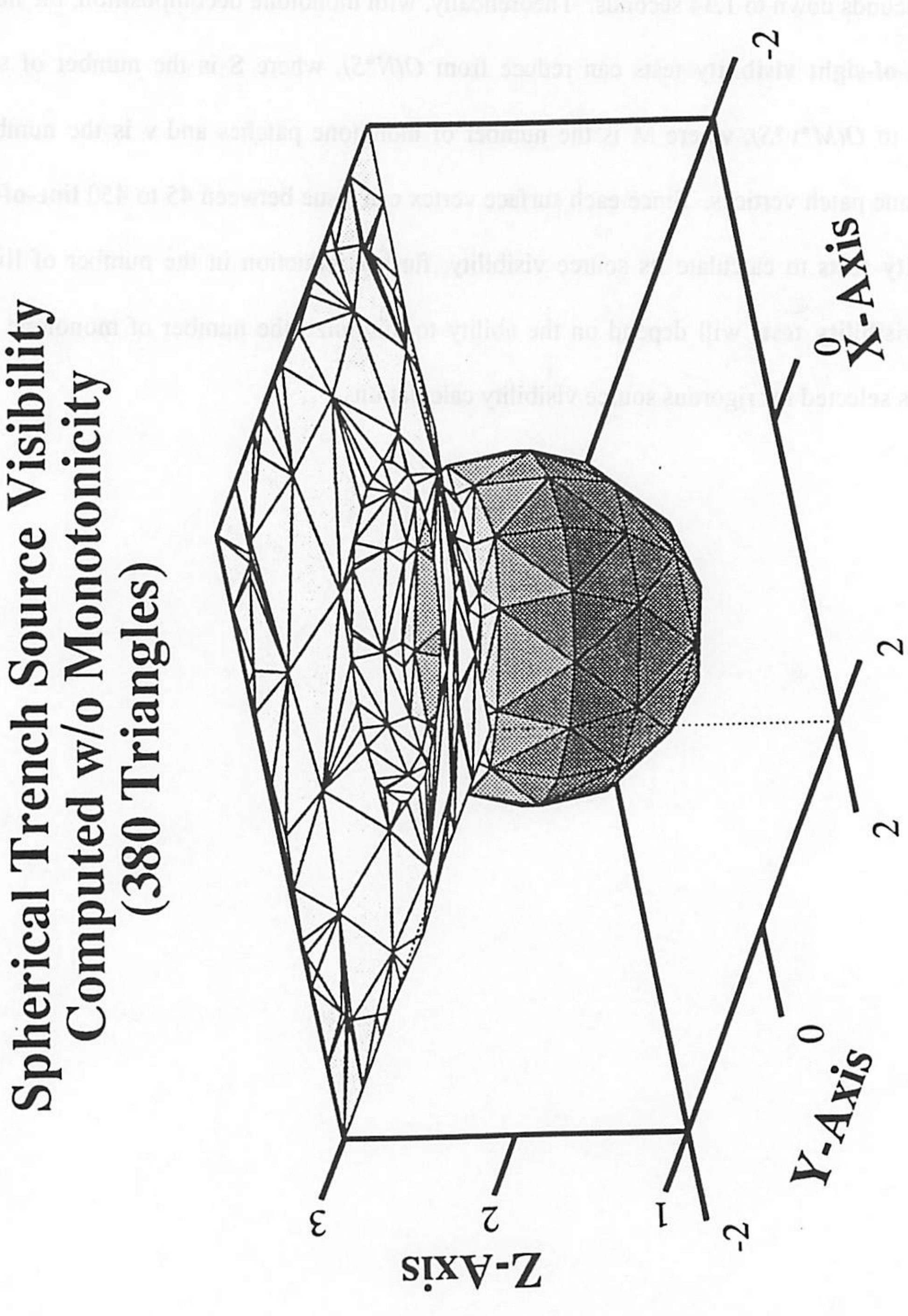


Figure 7.21

RHW – UCB TCAD

Spherical Trench Source Visibility Computed with Monotonicity (380 Triangles)

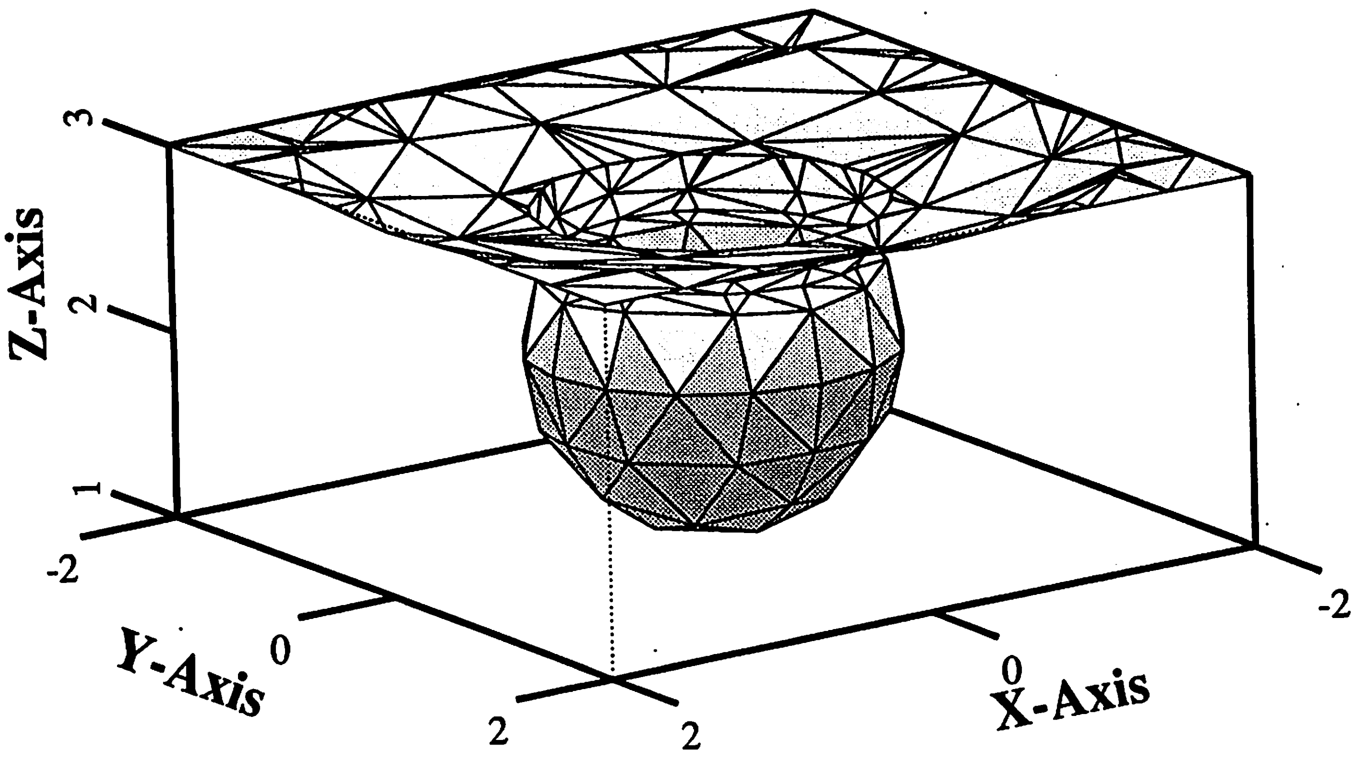
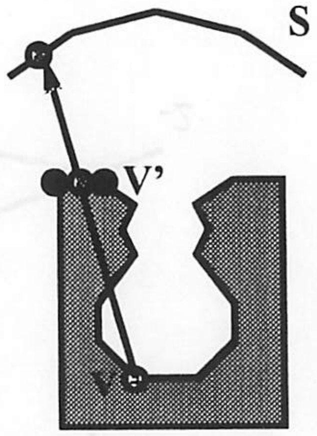


Figure 7.22

Table 7.2a

Computing Spherical Trench Source Visibility with and w/o Monotone Decomposition

{ IBM RS/6000, Model 530, 32 MB RAM }



# of Line–of–Sight Test Calls Comparison

Number of Triangles	With Monotone	No Monotone
380	7,807	23,224
904	8,176	75,554

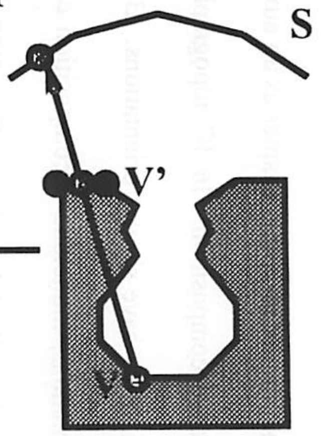
Memory Consumption Comparison

Number of Triangles	With Monotone	No Monotone
380	6,244 kB	5,672 kB
904	8,352 kB	7,072 kB

RHW – UCB TCAD

Computing Spherical Trench Source Visibility with and w/o Monotone Decomposition

{ IBM RS/6000, Model 530, 32 MB RAM }



CPU Time Comparison

Number of Triangles	Monotone Decomp.	With Monotone	No Monotone
380	2.83 sec	1.24 sec	3.07 sec
904	11.78 sec	1.34 sec	9.30 sec

Table 7.2b

7.9. CONCLUSIONS

Monotone decomposition was introduced in this chapter as an auxiliary data organization scheme for large grain surface decomposition. In IC topography simulation, large numbers of locally connected facets often have similar orientations. By bin sorting locally connected facets with similar orientations, monotone decomposition can easily partition a simulated surface into large grain monotone patches. Using monotone decomposition, a surface advance with global intra-surface collisions can be broken into a few well-behaved monotone patch advances. Void formations are transformed into overlaps between monotone deformation volumes. In other words, monotone decomposition efficiently focuses the robustness and power of merge operations where it is most needed.

Two types of monotone decomposition algorithms were described in this chapter: Greedy monotone decomposition and Directed monotone decomposition. The 2D **greedy monotone decomposition** algorithm incrementally groups each surface facet into a monotone chain in *O(N)* time. A monotone chain is terminated when the algorithm encounters a surface facet that fails to monotonically extend the chain. By locally maximizing the number of surface facets in each monotone chain, greedy monotone decomposition attempts to minimize the number of monotone chains. However, by not including the local z-axes of IC topographical features, the algorithm also tends to asymmetrically decompose axial symmetric features.

The **2.5D Isotropic Deposition Experiment** demonstrated the advantages and limitations of greedy monotone decomposition. For a 2D key hole trench with 48 facets, greedy monotone decomposition reduced the number of IBM Geometry Engine merge operations from 48 down to 2, and the total CPU time used by these operations from 22.42 seconds down to 1.04 seconds. However, as shown in Section 7.5, greedy monotone decomposition asymmetrically partitioned the surface at the lower right corner of the key hole trench.

To symmetrically decompose 3D IC topographical features, the 3D **directed monotone decomposition** algorithm supported two schemes for partitioning facet orientations symmetrically about the local *z*-axes of IC topographical features. According to the sign of the facet normal *x*-component, the **x-cut monotone decomposition** algorithm classifies each facet as one of **three** categories: **Left-pointing, Right-pointing**, and **Horizontal**. Based on the signs of the facet normal *x*- and *z*-components, the **xz-cut monotone decomposition** algorithm classifies each facet as one of **eight** categories by further identifying each facet as **Upward-pointing, Downward-pointing**, or **Vertical**. Despite using coarse partitions of facet orientations, the 3D directed monotone decomposition algorithm can symmetrically decompose individual features in a complex topography, such as periodic spherical trenches.

The **3D Isotropic Deposition Experiment** demonstrated the need for large grain surface decomposition methods, such as directed monotone decomposition, in 3D boundary deformation. The estimated CPU time reduction showed that monotone decomposition makes it feasible to use IBM Geometry Engine **merge** operations in 3D boundary deformation. For a 3D key hole trench with 896 triangles, the number of **merge** operations was reduced from 895 down to 8. Correspondingly, using CPU time data from the **Localized Deformation Test** in Section 6.6, the total CPU time used by IBM Geometry Engine **merge** operations was estimated to be reduced from about 5,370 seconds down to about 48 seconds.

The **3D Source Visibility Experiment** demonstrated the use of directed monotone decomposition to reduce the number of **line-of-sight visibility tests** in SAMPLE-3D source visibility computation. In addition, the test pointed out the need for special-purpose shading interpolation algorithms that exploit the facet locality and orientation similarities inherent in monotone patches. For a hemispherical source with 450 points, and a spherical trench surface with 904 triangles, the number of SAMPLE-3D **line-of-sight visibility tests** was reduced from 75,554 down to 8,176. Correspondingly, the total CPU time used by SAMPLE-3D **line-of-sight visibility tests** was reduced from 9.30 seconds down to 1.34 seconds. Since each surface vertex can issue between 45 to 450 **line-of-sight visbility tests** to calculate its source visibility, further reductions in the number of tests will depend on the ability to minimize the number of monotone patch vertices selected for rigorous source visibility calculation.

REFERENCES FOR CHAPTER 7

[1] R.H. Wang, M.S. Karasick, and A.R. Neureuther, Computational evaluation of three-dimensional topography process simulation components. International Workshop on VLSI Process and Device Modeling (VPAD), Kyoto, Japan, May 1993, pp. 95-96.

[2] R.H. Wang, and A.R. Neureuther, Efficient and innovative use of three-dimensional geometry services in IC topography simulation. International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA), Taipei, Taiwan, ROC, June 1995.

[3] M. Karasick, D. Lieber, Schemata for interrogating solid boundaries. ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[4] E.W. Scheckler, A.R. Neureuther, Models and algorithms for three-dimensional topography simulation with SAMPLE-3D. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Feb. 1994, vol.13, (no.2):219-30.

[5] Preparata and Shamos, *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[6] M.S. Karasick, On the representation and maniuplation of rigid solids. Ph.D. Thesis, McGill University (also Cornell University TR 89-976), 1989.

[7] S. Hamaguchi, M. Dalvie, R.T. Farouki, S. Sethuraman, A shock-tracking algorithm for surface evolution under reactive-ion etching. Journal of Applied Physics, 15 Oct. 1993, vol.74, (no.8):5172-84.

[8] W.G. Oldham, S.N. Nandgaonkar, A.R. Neureuther, M. O'Toole, M. A general simulator for VLSI lithography and etching processes. I. Application to projection lithography. IEEE Transactions on Electron Devices, April 1979, vol.ED-26, (no.4):717-22.

[9] W.G. Oldham, A.R. Neureuther, J. L. Reynolds, S.N. Nandgaonkar, and others. A general simulator for VLSI lithography and etching processes. II. Application to deposition and etching. IEEE Transactions on Electron Devices, Aug. 1980, vol.ED-27, (no.8):1455-9.

[10] J.P. McVittie, J.C. Rey, A.J. Bariya, M.M. IslamRaja, and others. SPEEDIE: a profile simulator for etching deposition. Proceedings of the SPIE - The International Society for Optical Engineering, 1991, vol.1392:126-38.

[11] Thurgate, T. Segment-based etch algorithm and modeling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Sept. 1991, vol.10, (no.9):1101-9.

[12] C.H. Sequin, Computer simulation of anisotropic crystal etching. Sensors and Actuators A (Physical), Sept. 1992, vol.A34, (no.3):225-41.

[13] I.V. Katardjiev, Simulation of surface evolution during ion bombardment. Journal of Vacuum Science & Technology A July-Aug. 1988, vol.6, (no.4):2434-42.

[14] D.S. Ross, Ion etching: an application of the mathematical theory of hyperbolic conservation laws. Journal of the Electrochemical Society, May 1988, vol.135, (no.5):1235-40.

[15] J. Sefler, 3D Surface Modeling Utilities for use in TCAD, MS Thesis, UC Berkeley, October 28, 1995.

[16] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, 2nd Ed., Addison-Wesley Publishing Co., 1987.

CHAPTER 8

AUXILIARY DATA STRUCTURES FOR IC TOPOGRAPHY PROPAGATION TRACE-BACK

8.1. INTRODUCTION

Topography propagation plays a dominant role in determing the shapes and dimensions of IC topographical features. From the viewpoint of process integration, topography propagation is the result of complex interactions between process steps and layout masks. Section 8.2 introduces **IC topography propagation trace-back** as a new TCAD functionality that allows users to start with a faulty topographical feature, and trace back the process steps or layout masks that might have caused it. Since process flow and layout dependencies may be attached to temporary masking layers, such as resist layers, topography propagation trace-back requires the support of auxiliary data structures that propagate dependencies down to topographical features. This chapter introduces two auxiliary data structures for supporting topography propagation trace-back. The first is process history tagging, which has been prototyped using the IBM Geometry Engine [1] and SIMPL System 6 [2]. The second is process interaction tracking, which is being proposed for implementation in next generation 3D geometry servers.

Auxiliary data structures for IC topography propagation trace-back can be implemented by semantically extending **solid model attribution** services to relate geometry components to process steps and layout masks. Section 8.3 discusses tradeoffs and issues involved in designing these semantic extensions. **The key tradeoff in the design of semantic extensions for topography propagation trace-back is the maintenance of accurate process flow and layout dependencies on geometry components at a minimal storage cost.** Three design issues arise from addresing this tradeoff. First, there is the need to determine the largest granularity of tagged geometry components which yields accurate dependencies propagation. To minimize storage cost, a compact representation of component dependencies need to be defined. Finally, process specific dependencies propagation utilities have to be implemented to compensate for incomplete geometry server attribute propagation.

To study these design issues, this chapter describes two examples of semantic extensions: Process history tagging and Process interaction tracking. **Process history tagging** is a straightforward extension of solid model attribution services which involves directly tagging geometry components with process flow and layout dependencies. In a 3D IC topography, topographical features can be represented as volumes, which have large component granularity. Therefore, it may be feasible to use **volume tagging** as a storage efficient method for maintaining accurate dependency information. To study volume tagging, Section 8.4 describes a 2.5D process history tagging data structure which has been implemented using the IBM Geometry Engine [3]. As compact representations for process flow and layout dependencies, the data structure uses SIMPL System 6 [4] process step and layout mask id's as solid model tags. For implementation using IBM Geometry Engine attribute propagation, the data structure includes rules to force additional dependencies propagation after SIMPL lithography, etching, and deposition simulation.

While volume-based process history tagging can minimize storage cost, the coarse granularity inherent in volume tagging may cause the attachment of false dependencies. To evaluate process history tagging, Section 8.5 introduces **topography propagation trace-back**. Topography propagation trace-back walks through the tags on each IBM Geometry Engine volume, and reports the SIMPL process steps and layout masks which influenced the volume's creation. In Section 8.5, topography propagation trace-back is demonstrated using the **Two Spacer Trace-Back Experiment**, which simulates a complex topography containing spacers cut from two separately deposited layers.

For next generation 3D geometry servers, Section 8.6 proposes **Process interaction tracking** as an accurate and storage efficient auxiliary data structure for IC topography propagation trace-back. The power of process interaction tracking derives from recognizing **process interaction** as a compact representation of process flow and layout dependencies, as well as an unambiguous criterion for partitioning geometry components. With minor modifications to its boolean set operations, a geometry server can incrementally compile a tree (or list) of process interactions, and bin sort geometry components according to the process interactions which created them. Process interaction tracking can be efficiently implemented as a table of process flow and layout dependencies indexed by process interactions, and a set of tags that link each geometry component to the underlying process interaction tree. By traversing tags in the process interaction tracking auxiliary data structure, any geometry component can readily use its creating process interaction to look up the associated process flow and layout dependencies.

8.2. IC TOPOGRAPHY PROPAGATION TRACE-BACK

IC topography propagation trace-back is a new TCAD functionality which enables IC technologists to start with a faulty topographical feature, and trace back the process steps or layout masks that might have caused it. Figure 8.1 depicts two examples of topography propagation effects, and the types of questions that topography propagation trace-back should be capable of answering. In the first example, shown on the left of Figure 8.1, a metal stringer was created as a result of topography propagation from the polysilicon line. In this basic example, a topography propagation trace-back should be capable of showing that the shape of the metal stringer is dependent on the process steps and layout masks which patterned the polysilicon line, and coated the line with dielectric material.

The second example, illustrated on the right of Figure 8.1, shows a complex IC topography which contains two spacer materials. In the right of Figure 8.1, the first spacer (**Spacer 1**) material is denoted by horizontal line patterns, and the second spacer (**Spacer 2**) material is denoted by checkered line patterns. In this example, **Spacer 1** material was created by a simple etch-back process that used the polysilicon line as an etch stop. **Spacer 2** material contains four features, two of which are the result of topography propagation from **Spacer 1** material. In this case, a topography propagation trace-back should show that the stringer and the rightmost (compound) spacer in the **Spacer 2** material are the results of topography propagation from **Spacer 1** material. On the other hand, the trace-back should show that the leftmost spacer and the upper spacer in the **Spacer 2** material were only influenced by an etch-back process that used the oxide layer as an etch stop.

To facilitate IC topography propagation trace-back, auxiliary data structures are needed to link geometry components back to the process steps and layout masks which created them.

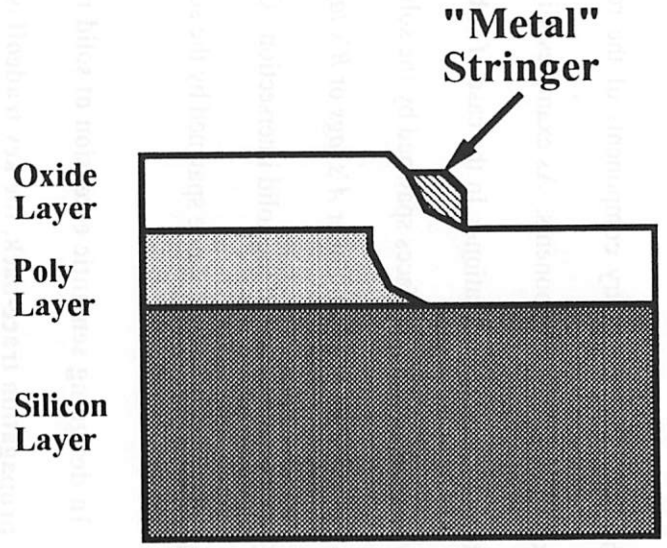
Previous work on coupling geometry to process flow and layout showed that such auxiliary data structures would represent an important evolutionary step in this new research area. For example, Berkeley's SIMPL-DIX system [3] provides a **Hunch** facility, which simulates and highlights device topographies near areas which contain layout design rule violations. The Hunch facility can be used to verify whether a layout design rule violation leads to a fatal electrical fault on the device topography. However, it does not allow users to trace back faulty topographical features.

As another example, in MIT's MEMCAD system [4] and ETH Zurich's ISE TCAD system [5], a partial process history is generated along with wafer geometry components. The process history includes parameters of additive process steps, such as the type of deposition (e.g. thermal oxidation versus oxide CVD). In these systems, this information is used to automate the generation of material properties for electromechanical or device simulation. However, since it does not record the effects of subtractive process steps, such as lithography and etching, this process history cannot sufficiently support topography propagation trace-back.

Auxiliary data structures for IC topography propagation trace-back can be implemented by semantically extending geometry servers to leave imprints of process steps or layout masks on geometry components. General-purpose geometry servers provide **solid model attribution services**, which can be used to define process flow and layout dependencies as solid model attributes. Services are also provided to attach and propagate these attributes onto geometry components during boolean set operations. Consequently, process flow and layout dependencies of geometry components can be incrementally maintained as the IC topography evolves. In the next section, issues which arise from the semantic extension of solid model atttribution services will be discuss in more detail.

Types of questions that IC topography proopagation trace–back should be capable of answering.

What are the process steps and layout masks that influenced the shape of the metal stringer?



Which features of "Spacer 2" material are the result of topography propagation from "Spacer 1" material?

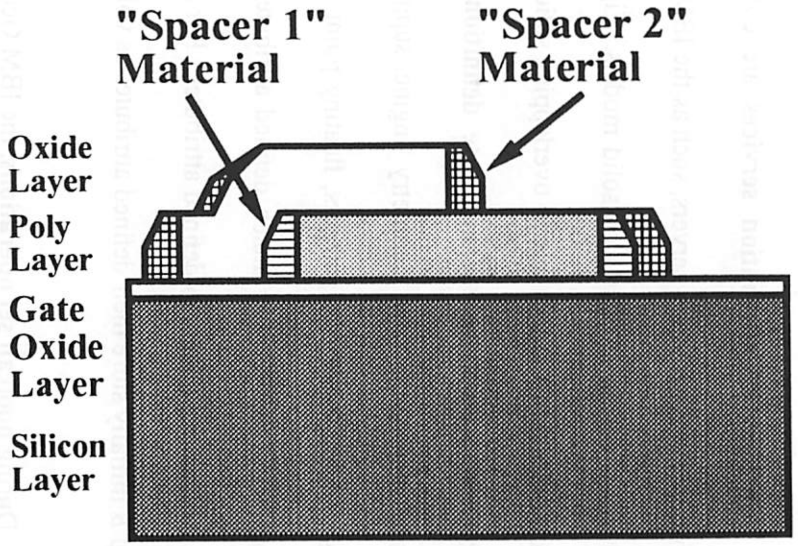


Figure 8.1

RHW – UCB TCAD

8.3. SEMANTIC EXTENSIONS OF SOLID MODEL ATTRIBUTION SERVICES

Solid model attribution services are convenient facilities originally provided by general-purpose geometry servers, such as the IBM Geometry Engine, to attach and propagate mechanical design attributes on solid models. Figure 8.2 illustrates the application of solid model attribution services to two overlapping solids: Solid *A* and Solid *B*. Basic solid model attribution services include **attribute definition** and **attribute tagging**. Most geometry servers, such as the IBM Geometry Engine, support the definition of the following attribute types: Character strings, integers, floating point numbers, and object pointers (i.e. memory addresses). To efficiently attach defined attributes, geometry components can maintain **tag lists**, or lists of pointers to defined attributes. For example, Figure 8.2 shows that Solid *A* and Solid *B* initially store their defined attributes as volume tag lists: *A's tags* and *B's tags*.

During **boolean set operations**, the IBM Geometry Engine and similar geometry servers perform a solid model attribution service known as **self attribute propagation**. Self attribute propagation tags geometry components of the resulting solid according to the input solids which spawned the components. As examples, Figure 8.2 illustrates self attribute propagation for two boolean set operations. In the case of **subtraction**, the volume of Solid *A-B* contains only *A's tags*, while the faces spawned by the solid intersection contain both *A's tags* and *B's tags*. In the case of **inset**, either *A's tags* or *B's tags* are propagated onto the volumes of Solid *A+B* which are outside of the solid intersection. On the other hand, both *A's tags* and *B's tags* are propagated onto the volume spawned by the solid intersection.

In designing semantic extension of solid model attribution services for IC topography propagation trace-back, the key tradeoff is the maintenance of accurate process flow and layout dependencies on geometry components at a minimal storage cost. Several issues arise from addressing this design tradeoff: 1) Determining the largest granularity of tagged geometry components which yields accurate dependencies propagation. 2) Defining a compact representation of process flow and layout dependencies which minimizes storage cost. 3) Building process specific dependencies propagation utilities to compensate for insufficient server attribute propagation.

For accurate maintenance of process flow and layout dependencies, it may be necessary to tag small grain components, such as faces in 3D and edges in 2D. Certain IC topographical features, such as trenches and voids, are represented by empty spaces. Since **boundary representation (B-Rep)** volumes cannot be used to represent empty spaces, **volume tagging** cannot be used to maintain the dependencies necessary for topography propagation trace-back of trenches and voids. Figure 8.3 depicts this scenario for a silicon trench etched using an oxide mask. As shown in Figure 8.3, while the oxide mask's dependencies can be attached to its volumes, the silicon trench's dependencies must be attached to its bounding faces.

Another problem with volume tagging is the coarse granularity of tagged geometry components may lead to attachment of false dependencies. When dependencies propagation is confined to 3D volumes or 2D faces, all topographical features cut from the deposited layer will contain the dependencies attached on the deposited layer. However, as will be demonstrated in Section 8.5, some of these features may actually sit on top of the flat portions of the underlying topography. Therefore, in this case, the coarse component granularity of volume tagging has led to a contradicting situation, in which component dependencies suggest that topography propagation has influenced the shape of features sitting on planar surfaces.

On the other hand, the practical minimum granularity of tagged geometry components is limited by the storage efficiency of process flow and layout dependencies representations. For example, at first glance, process step and layout mask id's seem to be logical choices for compact representations of process flow and layout dependencies. However, since these id's need to be explicitly tagged at each geometry component, process step and layout mask id's are not a storage efficient dependencies representations. In a typical IC process technology, a process flow may have about 100 process steps, and a layout may have about 20 mask levels. Therefore, each geometry component can potentially contain about 120 solid model tags. In a typical 3D IC topography, there may be about 10,000 vertices, edges, or faces. Hence, if process step and layout mask id's are used as dependencies representation for small grain geometry component tagging, 1,200,000 solid model tags may need to be stored. Assuming each tag is implemented as a bit mask, a total of 1.2×10^6 bits / 8 bits/byte = 167 Kbytes are required to store the solid model tags.

If process step and layout mask id's are chosen as dependencies representations, volume tagging would be the only practical method for semantically extending solid model attribution services. In a simulated IC topography, the number of topographical features is typically fewer than 20. Assuming each volume contains 120 solid model tags, this translates to a total of 2,400 solid model tags. Therefore, the storage cost would only be 2400 bits / 8 bits/byte = 0.3 Kbytes. Consequently, despite its obvious shortcomings in terms of attribute propagation accuracy, volume tagging is further evaluated in Section 8.4 as a practial semantic extension of solid model attribution services.

Finally, to compensate for incomplete self attribute propagation in general-purpose geometry servers, special-purpose utilities are needed to force propagation of deformation volume dependencies. As an example that requires forced dependencies propagation, Figure 8.4 depicts the geometry update after a silicon trench etching step. Some general-purpose geometry servers, such as the IBM Geometry Engine, do not propagate volume tags onto faces during subtraction. Therefore, as shown in the left of Figure 8.4, a special-purpose utility may be needed to force a propagation of etching volume dependencies before substraction. As a result of this work around, the right of Figure 8.4 shows that volume-based self attribute propagation can then be used to maintain accurate dependencies during etching geometry update.

Self Attribute Propagation during Boolean Set Operations

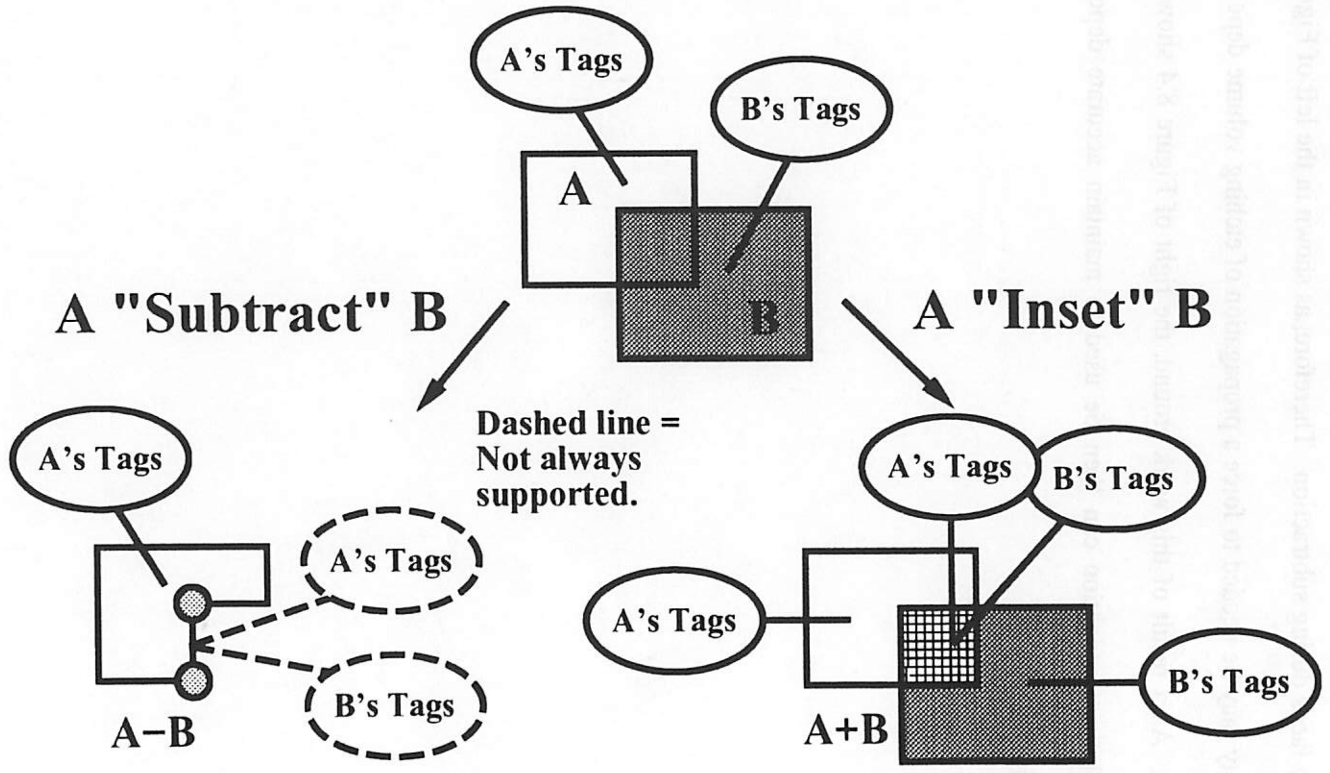


Figure 8.2

Issue 1) Need to tag small grain components for accurate dependencies propagation.

Oxide layer's depends can be maintained on volume.

Silicon trench is represented by empty space. Its depends must be tagged on bounding faces.

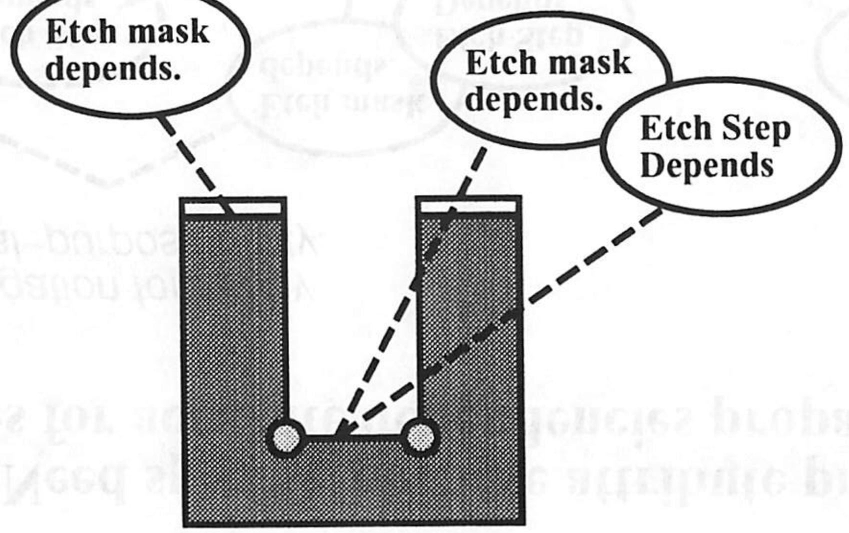


Figure 8.3

RHW – UCB TCAD

Issue 3) Need special–purpose attribute propagation utilities for accurate dependencies propagation.

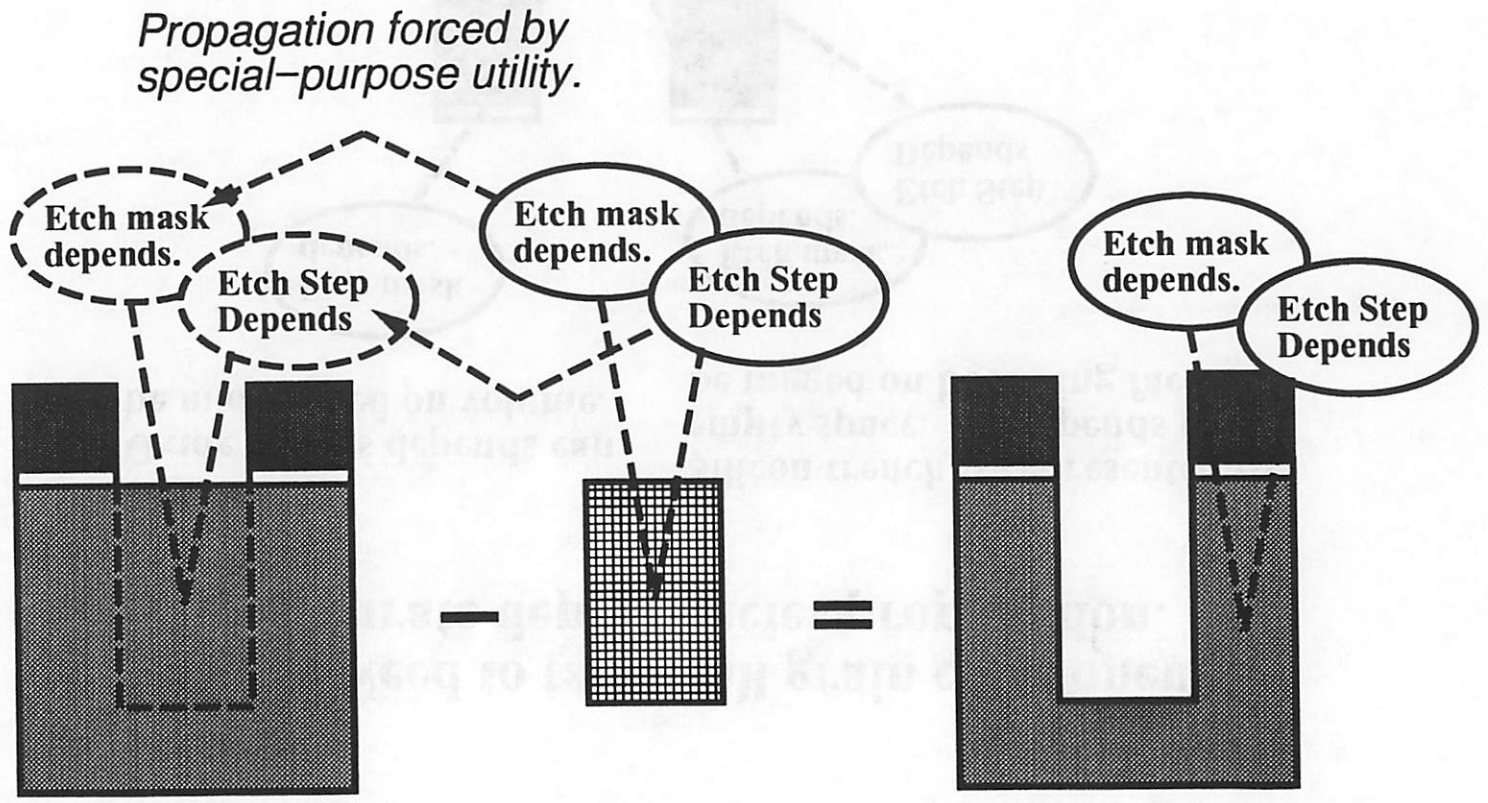


Figure 8.4

8.4. 2.5D VOLUME-BASED PROCESS HISTORY TAGGING

The 2.5D volume-based **process history tagging** data structure was first introduced as part of this thesis work in [6]. This data structure implements **volume tagging** by semantically extending IBM Geometry Engine solid model attribution services. The data structure uses SIMPL System 6 process step and layout mask id's as representations of process flow and layout dependencies. The data structure implementation involves two types of process specific utilities for maintaining accurate dependencies on IC topographical features (i.e. volumes in 3D and faces in 2D). These utilities are straightforward semantic extensions of the IBM Geometry Engine. First, a **tag conversion** utility is implemented to convert process specific attributes into deformation volume tags. Secondly, for process steps which cut into the inital topography, separate **depedencies propagation** utilities pass deformation volume tags down to the modified geometry components.

To streamline the conversion and propagation of deformation volume tags, process flow and layout dependencies have been grouped into the following types: 1) Direct processing dependencies, 2) Masking layer dependencies during subtractive process simulation, and 3) Non-planar underlayer dependencies during additive process simulation. The following paragraphs describe tag conversion and propagation rules for the three dependencies types. In particular, surface and material interface planarity is identified as an important filter for extraneous dependencies. To automate dependencies tracking during each simulation time step, these rules have been incorporated into BTU's geometry update utilities for lithography, etching, and deposition.

Direct processing dependencies identify the process steps and layout masks directly responsible for the creation or deformation of an IC topographical feature. Lithography is a process step that introduces only direct processing dependencies. Figure 8.5 illustrates tag conversion and propagation rules for direct processing dependencies generated by the lithography process. In Figure 8.5, the wafer states before and after the lithography update are shown on the left and right of the figure. As shown in Figure 8.5, deformation volume tags include the lithography process step id, and the applied layout mask id. Since IBM Geometry Engine only supports self attribute propagation, the lithography update utility is responsible for propagating deformation volume tags onto the resist layer before subtraction.

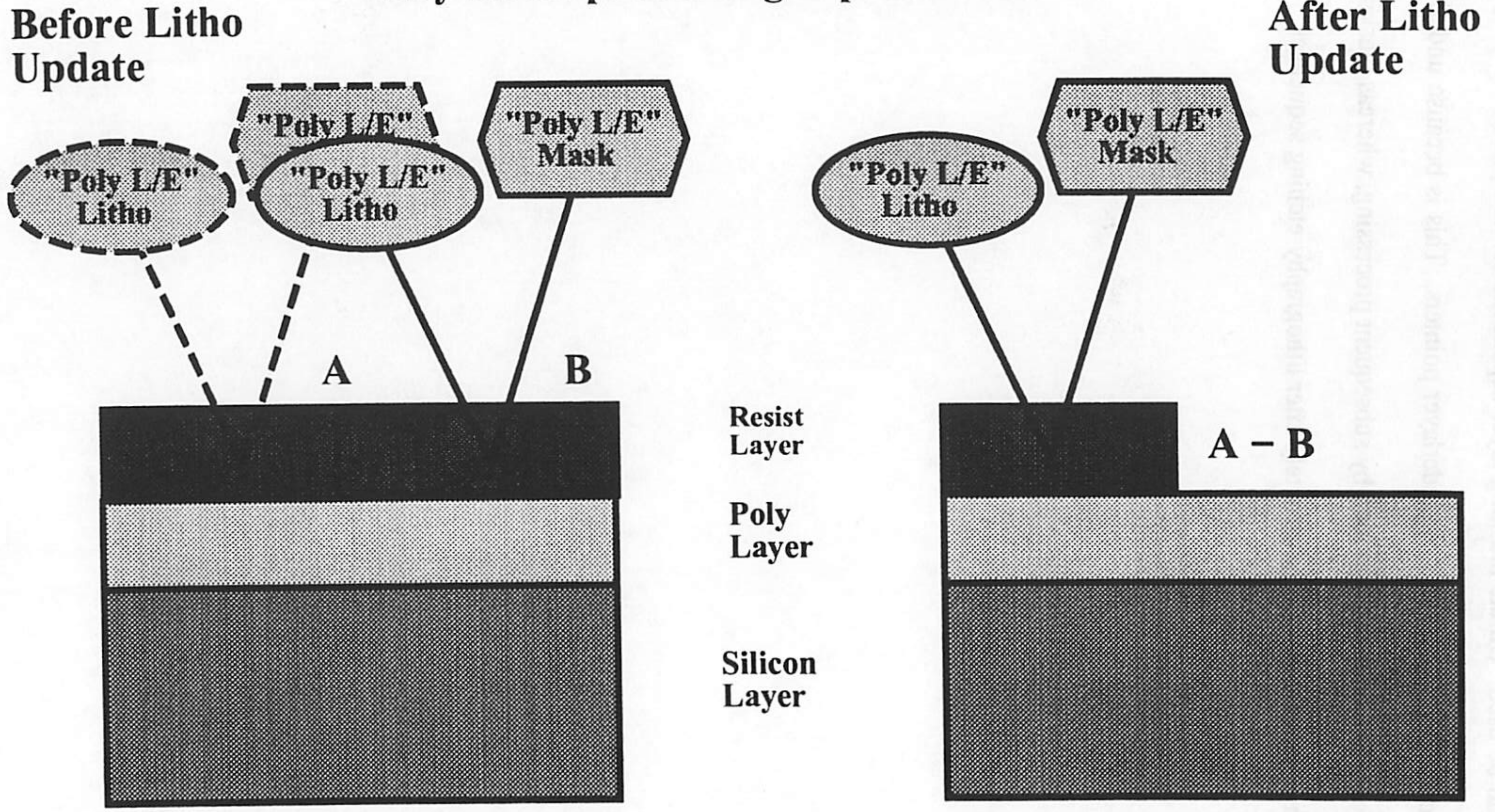
Masking layer dependencies record the process steps and layout masks that created or deformed the masking layer used in an etching step. Figure 8.6 illustrates tag conversion and propagation rules for masking layer dependencies generated by the etching process. In Figure 8.6, the wafer states before and after the etching update are shown on the left and right of the figure. As shown in Figure 8.6, deformation volume tags consist of the etch process step id, and masking layer tags. To identify **masking layers**, the etching update utility first partitions the wafer top surface into portions bounding various layers. Each top surface portion is checked for slope changes. Layers with top surface portions that exhibit slope changes exceeding a specified tolerance value are identified as masking layers. Since IBM Geometry Engine subtraction operation is also used in this case, the etching update utility is responsible for propagating deformation volume tags onto the etched layer before subtraction.

Non-planar underlayer dependencies account for the process steps and layout masks that created non-planar layers in the IC topography prior to a deposition step. Figure 8.7 illustrates tag conversion and propagation rules for non-planar underlayer dependencies generated by the deposition process. In Figure 8.7, the wafer states before and after the deposition update are shown on the left and right of the figure. As shown in Figure 8.7, deformation volume tags consist of the deposition process step id, and non-planar underlayer pointers. To identify **non-planar underlayers**, a procedure identical to the one for finding masking layers can be used. Unlike masking layer dependencies, non-planar underlayer dependencies can be succinctly represented by underlayer pointers. This is because underlayers, such as polysilicon lines, are usually buried by subsequent processing, whereas masking layers, such as resist layers, are removed immediately after lithography-etching sequences.

Figure 8.5

Direct Processing Dependencies

Lithography is a process that introduces only direct processing dependencies.



RHW – UCB TCAD

Masking Layer Dependencies

The etching deformation volume tags include both a reference to the etching step and tags inherited from the masking layer.

Before Etch Update and Ashing

Resist layer is masking Poly layer.

After Etching Update and Resist Removal

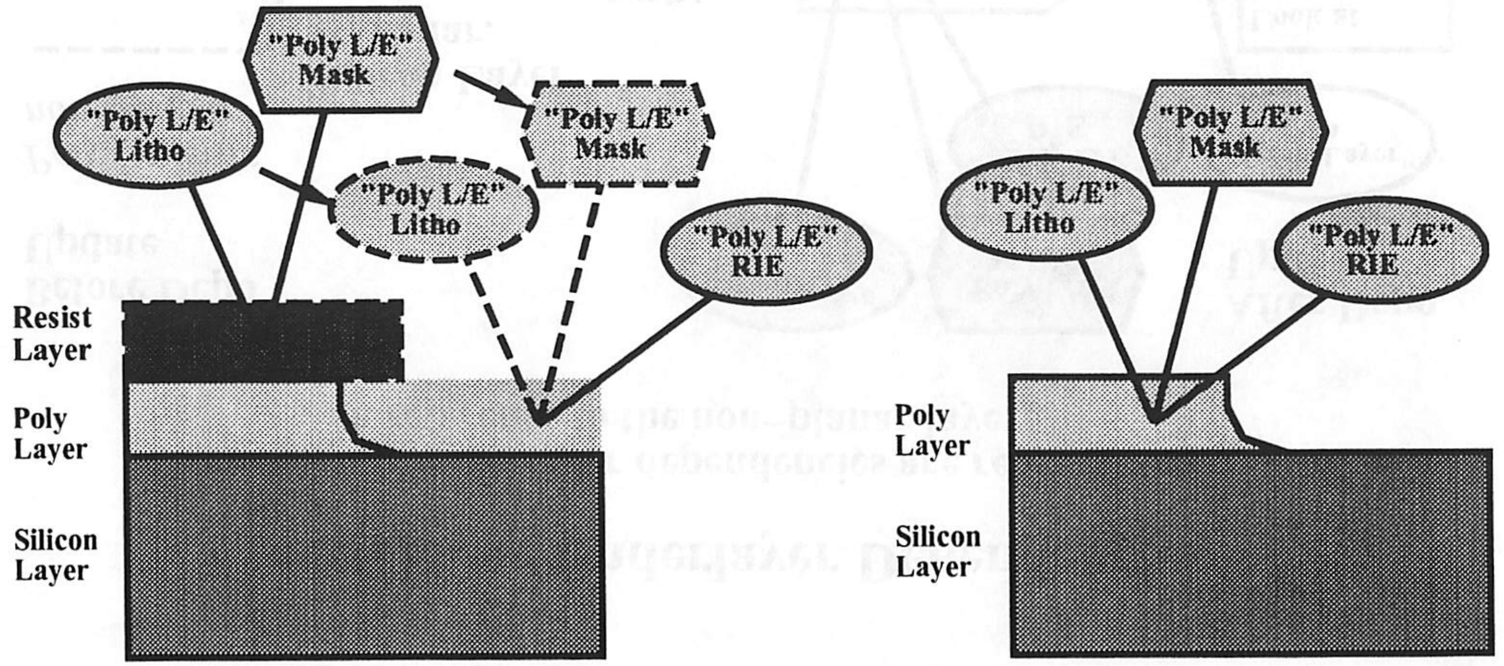


Figure 8.6

Non–Planar Underlayer Dependencies

Non–planar underlayer dependencies are represented by a reference to the non–planar layer.

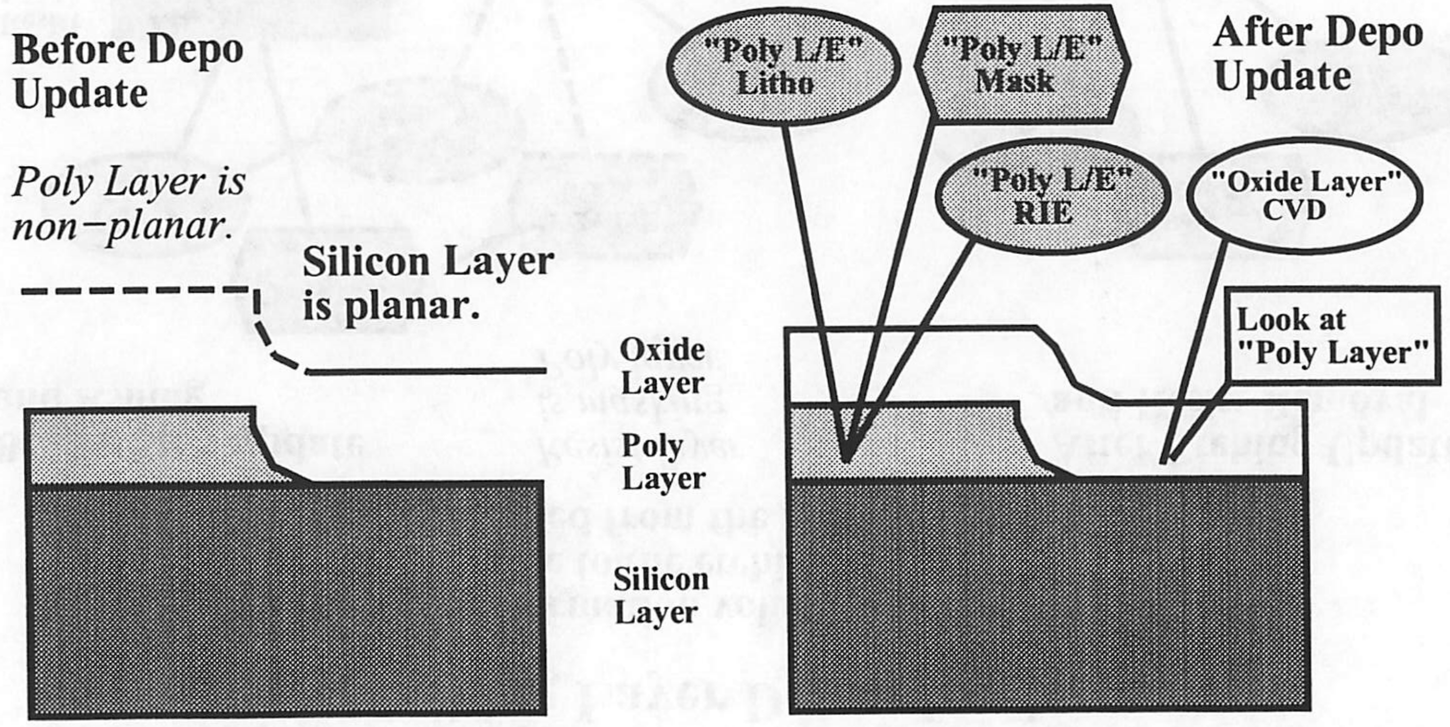


Figure 8.7

8.5. 2.5D VOLUME-BASED TOPOGRAPHY PROPAGATION TRACE-BACK

To evaluate the usefulness of process history tagging, the 2.5D volume-based **topography propagation track-back** data structure was first introduced as part of this thesis work in [6]. Topography propagation trace-back can expose deeply buried process flow and layout dependencies by tracing through process step id's, layout mask id's, and underlayer pointers attached to IC topographical features (i.e. volumes in 3D and faces in 2D). In Section 3.5, the simulation and trace-back of a metal stringer structure was described as an example of BTU simulation experiments. To illustrate the topography propagation trace-back functionality, Figure 8.8 lists the order in which the dependencies of the metal stringer are uncovered as "1" through "4", where "1" represents the first tag found. After tracing through two underlayer pointers (tags "2" and "3" in Figure 8.8), the trace-back function identifies the dependence of the metal stringer on lithography and etching of the polysilicon line.

As discussed in Section 8.3, the coarse component granularity associated with **volume tagging** can result in the attachment of false dependencies. This anomalous situation will be demonstrated here using results from the **Two Spacer Trace-Back Experiment**. This test began by using SIMPL to simulate a complex IC topography which contained spacers cut from two separately deposited layers. After each SIMPL topography process step, the resulting deformation volume was extruded and updated in the IBM Geometry Engine using the BTU geometry update utilities described in Section 8.4. Trace-backs were then performed on the features cut from the second deposited layer. As will be discussed below, the reported dependencies show that some of these features were erroneously tagged as strongly dependent on topography propagation.

The SIMPL process flow simulated by the **Two Spacer Trace-Back Experiment** is listed in Figure 8.9. Figure 8.10 shows the SIMPL layout and final cross section obtained from this process flow simulation. As listed in Figure 8.9, the major sequences in this process flow are: **Gate Oxide** layer deposition (thickness = 0.1 um, Step 1); **Poly 1** layer deposition and etching (thickness = 0.4 um, mask = POLY, Steps 2 through 4); **Spacer 1** material deposition and etch-back (thickness = 0.2 um, Steps 5 through 7); **Poly 2** layer deposition and etching (thickness = 0.4 um, mask = PLY2, Steps 8 through 10); **Spacer 2** material deposition and etch-back (thickness = 0.2 um, Steps 11 through 13); and **Metallization** (thickness = 0.4 um PSG and metal, mask = M1, Steps 14 through 19).

Topography propagation trace back of features in the **Spacer 2** material showed that volume tagging caused the attachment of extraneous dependencies on the lower left spacer and the upper right spacer. Figure 8.11 depicts the 3D topography simulated by the IBM Geometry Engine, and compares the dependencies reported for the upper right spacer (listed on the left), and the lower right spacer (listed on the right). As suggested in Figure 8.11, all features in the **Spacer 2** material contained identical dependencies. This is because the **Spacer 2** material was initially tagged as dependent on the **Spacer 1** material after deposition. After **Spacer 2** material etching, the etching update utility propagated all **Spacer 2** material dependencies onto all etched features. In fact, in the **Spacer 2** material, only the stringer and the lower right spacer truly resulted from topography propagation. Since the lower left spacer and the upper right spacer are situated atop flat portions of the **Gate Oxide** layer and **Spacer 1** material, they should only contain dependencies on **Spacer 2** material deposition and etch process steps, and perhaps the **Poly 2** line edges (e.g. the PLY2 mask).

Dependencies examined during Topography Propagation Trace–Back of metal stringer.

1, ..., 4 = Layer Tracing Order

"Stringer" Process and Mask Dependencies Tagged during Boundary Deformation

- ▒ **Inherited from Ashed Resist**
- **Attached by Poly Etch Process**
- ☐ **Attached by Oxide CVD Process**
- ▨ **Attached by Metallization Sequence**

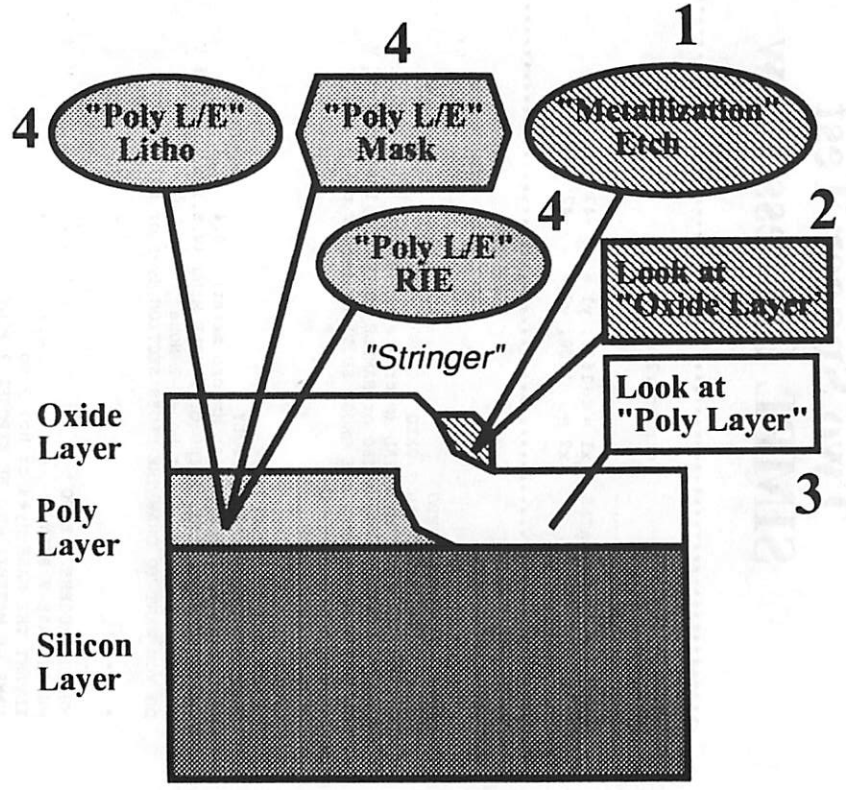


Figure 8.8

Two Spacer Test SIMPL Process Flow

```
******************************************************************************

LAYOUT FILE :          stringer2.cif

SUBSTRATE TYPE:
CUT-LINE COORDINATES : x1 =  -1600, y1 =     -47
                       x2 =   1600, y2 =     -47

******************************************************************************

* 1 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? OXID
THICKNESS OF THE MATERIAL (micro-meter) ? 0.1
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? V
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 2 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? POLY
THICKNESS OF THE MATERIAL (micro-meter) ? 0.4
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? V
DOPING (B, As, P, Sb or None) ? None
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 3 *

WHICH PROCESS ? EXPO
WHICH MASK ? POLY
INVERT THE MASK (yes or no) ? no
NAME OF MATERIAL TO BE EXPOSED ? POLY
NAME OF THE EXPOSED RESIST ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 4 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 5 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? NTRD
THICKNESS OF THE MATERIAL (micro-meter) ? 0.2
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? I
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 6 *

WHICH PROCESS ? ETCN
Etch Type:Isotropic, or Iso with Directional (1 or 10) ? 10
File containing etch rates ? ntrd.etch.mod
Etch accuracy (0:worst to 10:best) ? 10
Timestep in seconds ? 1
Number of steps ? 4
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 7 *

WHICH PROCESS ? ETCU
```

Figure 8.9

Two Spacer Test SIMPL Process Flow (Continued)

```
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 8 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? PLY2
THICKNESS OF THE MATERIAL (micro-meter) ? 0.4
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? I
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 9 *

WHICH PROCESS ? EXPO
WHICH MASK ? PLY2
INVERT THE MASK (yes or no) ? no
NAME OF MATERIAL TO BE EXPOSED ? PLY2
NAME OF THE EXPOSED RESIST ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 10 *

WHICH PROCESS ? DEVL
NAME OF THE LAYER TO BE DEVELOPED ? ERST
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 11 *

WHICH PROCESS ? DEPO
NAME OF THE MATERIAL ? NTD2
THICKNESS OF THE MATERIAL (micro-meter) ? 0.2
VERT, SPIN-ON, ISO, ANISO or SAMPLE MENU (V,S,I,A, or M) ? I
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 12 *

WHICH PROCESS ? ETCN
Etch Type:Isotropic, or Iso with Directional (1 or 10) ? 10
File containing etch rates ? ntrd.etch.mod
Etch accuracy (0:worst to 10:best) ? 10
Timestep in seconds ? 1
Number of steps ? 5
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

* 13 *

WHICH PROCESS ? ETCU
DO YOU WANT TO DRAW THE CROSS SECTION (yes or no) ? yes

WHICH PROCESS ? END
```

Figure 8.9 (Continued)

Two Spacer Test Layout and Cross Section

(SIMPL System 6 Simulation)

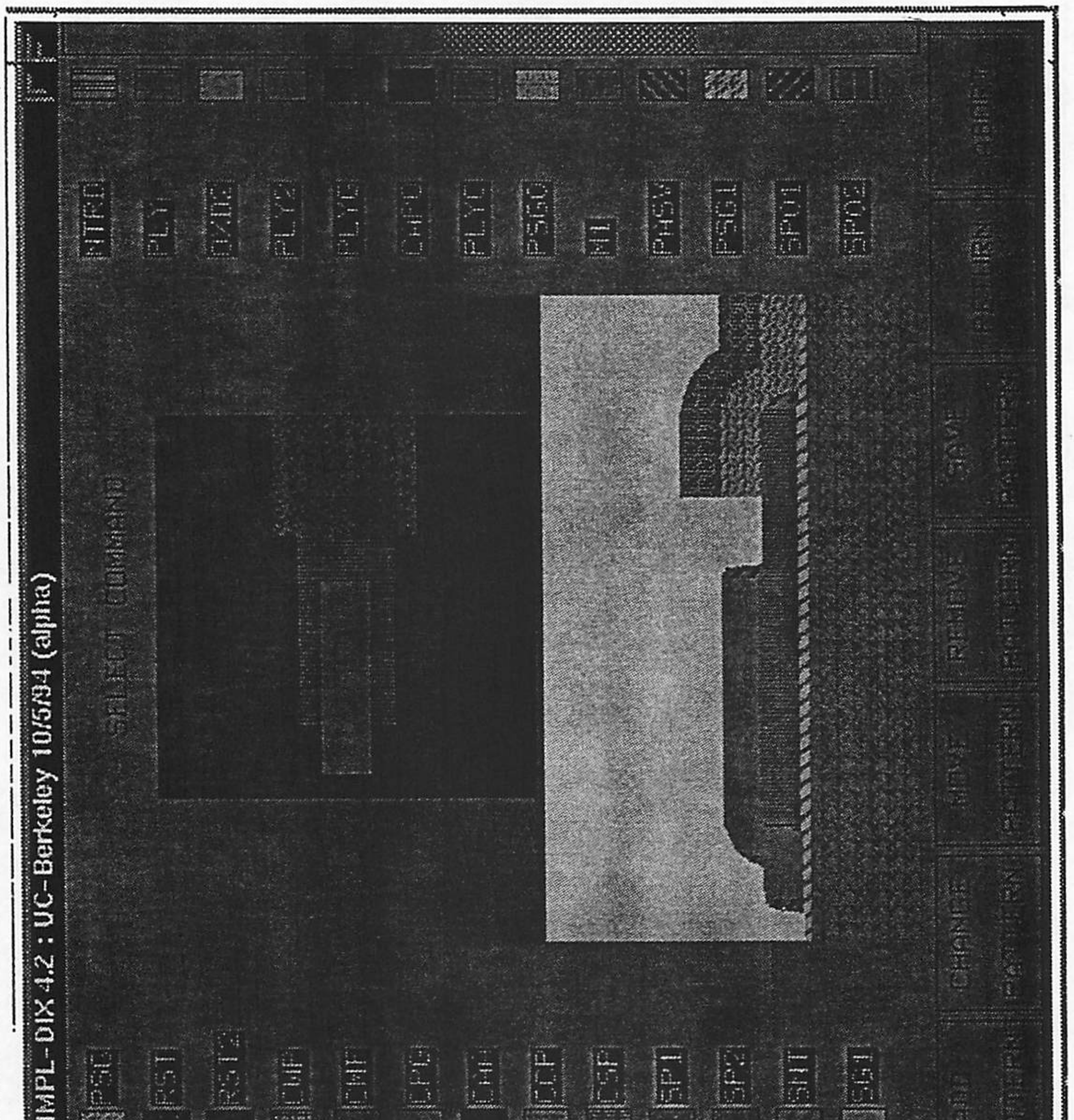
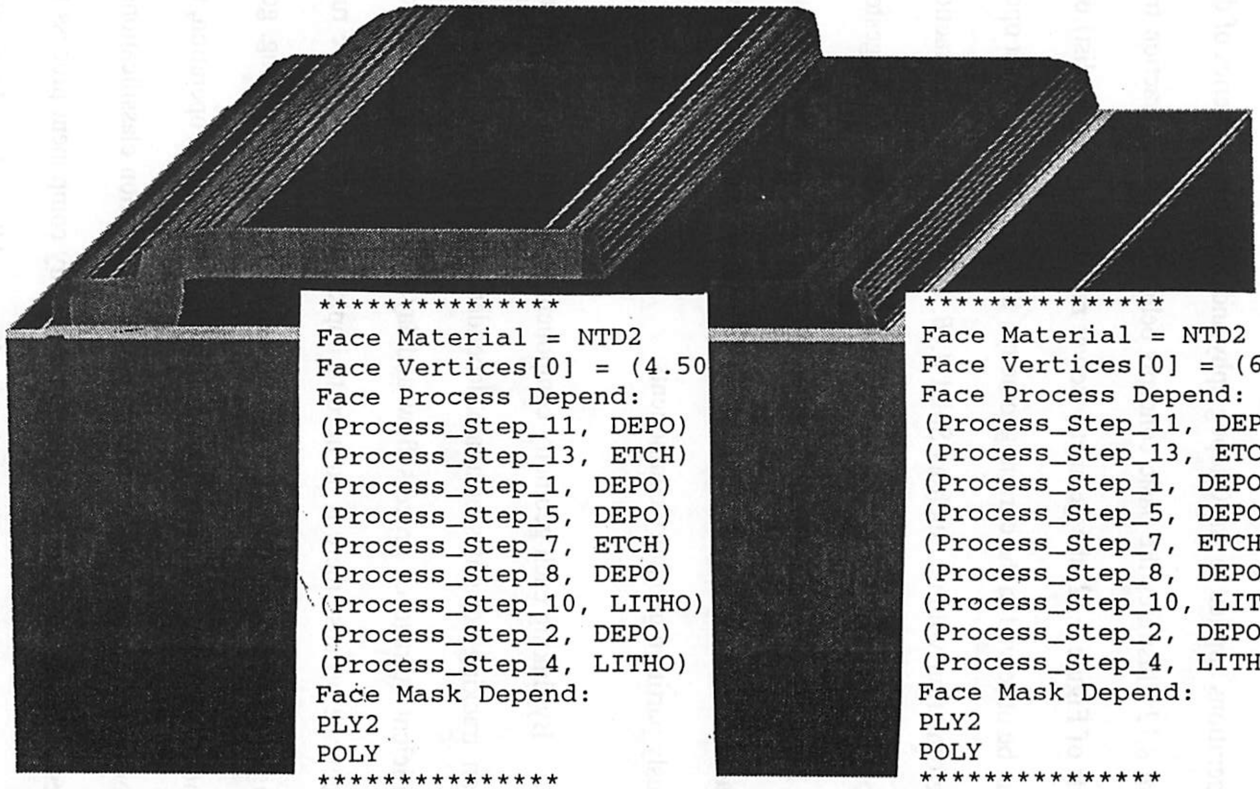


Figure 8.10

RHW – UCB TCAD

Two Spacer Test – Need to Tag Interfaces!

Lower right NTD2 spacer has correct mask dependencies.
Upper right NTD2 spacer should not depend on POLY mask.



```
***************
Face Material = NTD2
Face Vertices[0] = (4.50
Face Process Depend:
(Process_Step_11, DEPO)
(Process_Step_13, ETCH)
(Process_Step_1, DEPO)
(Process_Step_5, DEPO)
(Process_Step_7, ETCH)
(Process_Step_8, DEPO)
(Process_Step_10, LITHO)
(Process_Step_2, DEPO)
(Process_Step_4, LITHO)
Face Mask Depend:
PLY2
POLY
***************
```

```
***************
Face Material = NTD2
Face Vertices[0] = (6.67
Face Process Depend:
(Process_Step_11, DEPO)
(Process_Step_13, ETCH)
(Process_Step_1, DEPO)
(Process_Step_5, DEPO)
(Process_Step_7, ETCH)
(Process_Step_8, DEPO)
(Process_Step_10, LITHO)
(Process_Step_2, DEPO)
(Process_Step_4, LITHO)
Face Mask Depend:
PLY2
POLY
***************
```

Figure 8.11

RHW – UCB TCAD

8.6. PROPOSAL FOR 3D PROCESS INTERACTION TRACKING

With respect to volume-based process history tagging, this section introduces **process interaction tracking** as a more accurate and storage efficient data structure for supporting IC topography propagation trace-back. **Process interaction** refers to a series of boolean set operations applied to an IC topography and an associated series of deformation volumes. Figure 8.12 illustrates the basic concepts behind process interaction tracking. As shown on the left of Figure 8.12, the geometry server maintains a tree (or list) of process interactions that can be observed in the current geometry. Since each boolean set operation is associated with a deformation volume, such as Solid *A*, *B*, or *C*, a process interaction compactly represents a complete set of process flow and layout dependencies. As suggested by the arrows in Figure 8.12, each topography face is either the result of topography initialization, or can be traced back to a unique process interaction. Therefore, the tree of process interactions unambiguously partition geometry components.

By linking each geometry component to its creating process interaction, process interaction tracking separates accurate dependencies propagation on geometry components, from efficient storage of process flow and layout dependencies. Process interaction tracking is accurate because there is no need to implement process specific rules for additional attribute tagging or propagation. To support process interaction tracking, geometry servers need to be modified to incrementally update, after each boolean set operation, the process interaction tree (or list) and geometry component process interaction classifications. On the other hand, as a result of this software modification, geometry component process flow and layout dependencies can be automatically updated without additional (and presumably extraneous) attribute tagging or propagation.

Process interaction tracking also avoids the redundant storage of process step and layout mask id's on geometry components. From process flow and layout, a process interaction tracking auxiliary data structure can convert process interaction tree (or list) elements into a look-up table of interacting process steps and layout masks. For each geometry component, the auxiliary data structure can store a tag that links the geometry component to its creating process interaction. By traversing tags in the process interaction tracking auxiliary data structure, any geometry component can readily use its creating process interaction to look up the associated process flow and layout dependencies.

Process Interaction Tracking

Process Flow and Layout dependencies can be compactly represented by Process Interaction.

Geometry components can be unambiguously partitioned by Process Interaction.

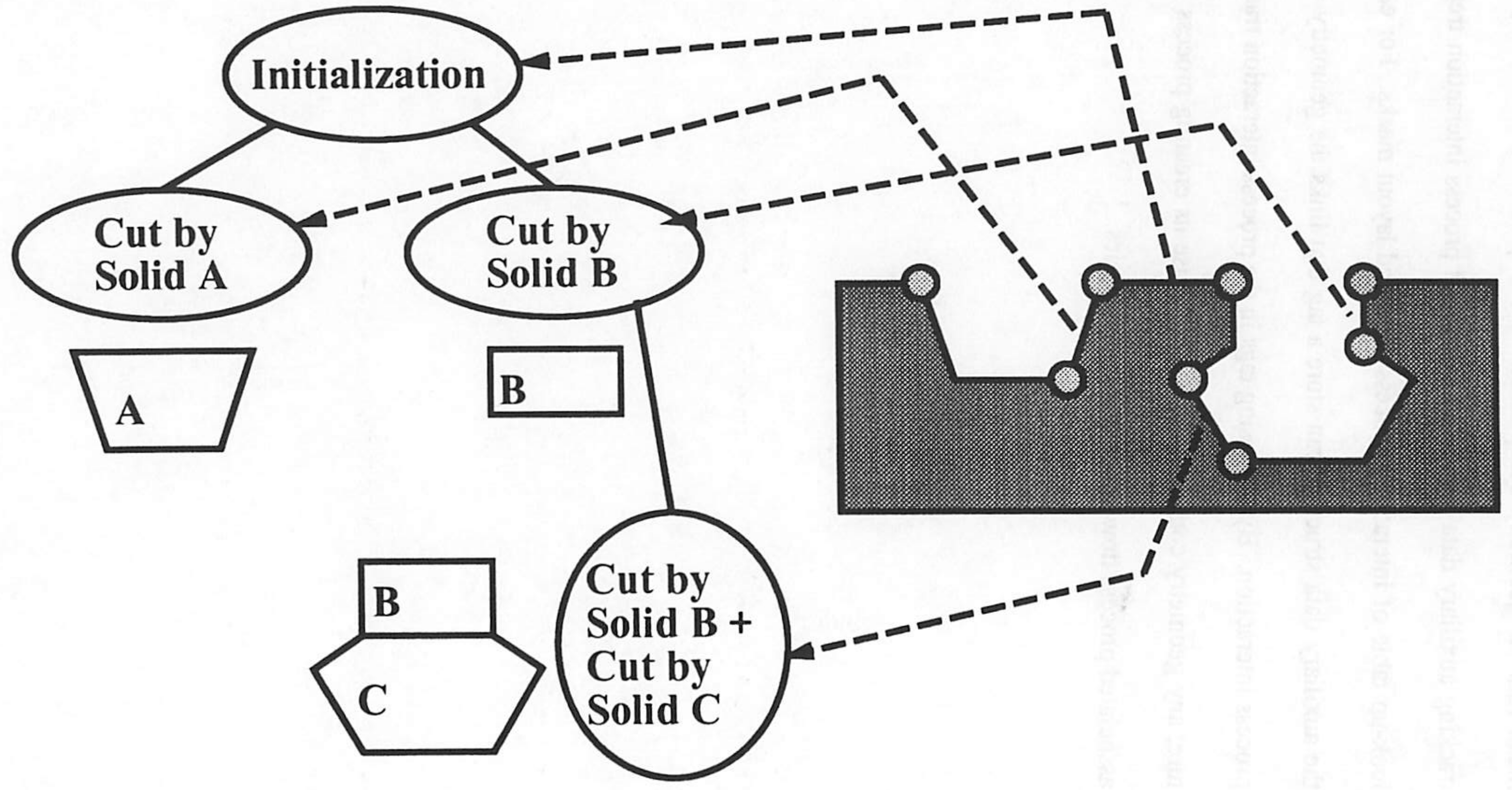


Figure 8.12

RHW – UCB TCAD

8.7. CONCLUSIONS

This chapter introduced a new TCAD functionality called **IC topography propagation trace-back**. This functionality enables IC technologists to start with a faulty topographical feature, and trace back the process steps and layout masks that might have caused it. Since process flow and layout dependencies may be attached to temporary masking layers, auxiliary data structures are needed to propagate dependencies down to topographical features. As a means of implementing auxiliary data structures for IC topography propagation trace-back. Section 8.2 suggested semantic extensions of **solid model attribution** services.

Section 8.3 explored issues in designing semantic extensions of solid model attribution services for IC topography propagation trace-back. For accurate maintenance of process flow and layout dependencies, it may be necessary to tag small geometry components, such as faces in 3D and edges in 2D. On the other hand, the practical minimum granularity of tagged geometry components is limited by the storage efficiency of process flow and layout dependencies representations. Finally, to compensate for **self attribute propagation** in general-purpose geometry servers, special-purpose utilities are needed to force propagation of deformation volume dependencies.

To further study volume tagging, Section 8.4 described a 2.5D **process history tagging** data structure. The data structure tagged IBM Geometry Engine volumes with SIMPL process step and layout mask id's. As discussed in Section 8.4, the primary advantage of volume-based process history tagging is its simplicity. The implementation of the 2.5D process history tagging data structure mainly involved building two auxiliary utilities which were straightforward wrappings of IBM Geometry Engine solid model attribution services. First, a **tag conversion** utility was developed to convert SIMPL process step and layout mask id's into IBM Geometry Engine solid model tags. Next, separate **dependencies propagation** utilities were developed to propagate tags onto IBM Geometry Engine volumes after SIMPL lithography, etching, and deposition simulations.

Due to the coarse granularity inherent in volume tagging, volume-based process history tagging frequently attaches false dependencies. This anomaly was demonstrated in Section 8.5 using **topography propagation trace-back** and the **Two Spacer Trace-Back Experiment**. The **Two Spacer Trace-Back Experiment** involved the simulation of a complex IC topography which contained spacers cut from two separately deposited layers. In this test, the layout masks were arranged such that some of the spacers cut from the second deposited layer rested on flat surfaces, and did not depend on process steps and layout masks previous to the second spacer etching sequence. However, due to the coarse granularity of volume tagging, the dependencies on the second deposited layer were propagated onto all spacers cut from it. As a result, trace-backs of the spacers resting on flat surfaces erroneously reported dependencies previous to the second spacer etching sequence.

With respect to volume-based process history tagging, Section 8.6 proposed **process interaction tracking** as a more accurate and storage efficient auxiliary data structure for next generation 3D geometry servers. By linking each geometry component to its creating process interaction, process interaction tracking separates accurate dependencies propagation on geometry components, from efficient storage of process flow and layout dependencies. Process interaction tracking is accurate because there is no need to implement process specific rules for additional attribute tagging or propagation. Instead, geometry servers are modified to automatically update the process interaction list (or tree) and geometry component dependencies. Process interaction tracking is storage efficient because process flow and layout masks dependencies are efficiently encoded by a single flag at each component. By traversing tags in the process interaction tracking auxiliary data structure, any geometry component can readily use its creating process interaction to look up the associated process flow and layout dependencies. Assuming each flag consumes a bit mask, 10,000 component flags only require about 10,000 / 8 = 1.25 Kbytes to store, which is two orders of magnitude less than the 167 Kbytes that could be used by small grain process history tagging.

REFERENCES FOR CHAPTER 8

[1] M. Karasick, D. Lieber, Schemata for interrogating solid boundaries. ACM Symposium on CAD and Foundations of Geometric Modeling, June 1991, pp. 15-25.

[2] D. Lee, Applying TCAD to Emerging Technologies, MS Thesis, UC Berkeley, UCB/ERL M95/38, May 20 1995.

[3] H.C. Wu, A.S. Wong, Y.L. Koh, E.W. Scheckler, and others. Simulated profiles from the layout- design interface in X (SIMPL-DIX). International Electron Devices Meeting. Technical Digest. San Francisco, CA, USA, 11-14 Dec. 1988. p. 328-31.

[4] S.D. Senturia, R.M. Harris, B.P. Johnson, S. Kim, and others. A computer-aided design system for microelectromechanical systems (MEMCAD). Journal of Microelectromechanical Systems, March 1992, vol.1, (no.1):3-13.

[5] P. Lamb, C. Hegarty, N. Hitschfeld, W. Fichtner, Generating solid models for VLSI process and device simulation. Proceedings of 1992 IEEE Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits: NUPAD IV, Seattle, WA, USA, 31 May-1 June 1992. pp. 175-80.

[6] R.H. Wang, and A.R. Neureuther, Efficient and innovative use of three-dimensional geometry services in IC topography simulation. International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA), Taipei, Taiwan, ROC, June 1995.

CHAPTER 9
CONCLUSIONS

9.1. SUMMARY OF FINDINGS

This thesis explores TCAD system organization for centralizing geometry services. Issues involved in the development, performance testing, and use of centralized geometry servers are investigated for 3D IC topography simulation. In many case, the issues have been identified and quantified through use of a prototype system based on linking SAMPLE-3D, SIMPL, and the IBM Geometry Engine as geometry servers through a hierarchically organized interface in the Berkeley Topography Utilities (BTU) system. The BTU geometry server interface consists of 41,000 lines of C++ code, and organizes geometric utilities and geometry services according to their input data granularity as Primitive, Auxiliary, and Aggregate.

This thesis initiates research in a new field of centralizing geometry services for 3D IC topography simulation, and makes contributions along several research fronts. First, this thesis links the performance of geometrical operations in 3D IC topography simulation to four essential geometry server constructs. These constructs can be implemented using conventional connectivity and spatial data structures, and special-purpose boolean set operations. The implementation of these constructs constitutes a necessary but not sufficient condition for efficient 3D IC topography simulation. This fact is demonstrated by comparing the theoretical performance of geometrical operations implemented with and without the constructs. Since the constructs are shown to be necessary for efficient 3D simulation, they may be used to screen potential servers.

Another contribution of this thesis is the definition of standardized performance tests. These tests are designed to mimic the stress placed on geometry servers during 3D IC topography simulation. The tests invoke geometrical operations on IC topographies at typical levels of physical detail and operation frequency. In other words, standardized performance tests can be used to evaluate the suitability of geometry servers for 3D IC topography simulation.

In implementing standardized performance tests, it is shown that the demand of IC topography simulation requires considerable augmentation of the standard interfaces to general-purpose geometry servers. About 10,000 lines of C++ code are likely necessary to perform relatively basic geometrical operations in 3D IC topography simulation with any solid modeling package. For example, about 12,000 lines of C++ code are required to interface the IBM Geometry Engine for performance testing. Out of these 12,000 lines, a significant part performs mundane geometry construction and data mapping tasks, such as constructing tiled initial topographies, extracting surface faces, triangulating polygonal faces, and constructing vertical deformation volumes.

To manage the large number of geometric utilities and services introduced by geometry servers, this thesis recommends a hierarchical server interface based on input data granularity of geometrical operations. The purpose for hierarchically organizing geometric utilities and services is to share codes that support data mapping, server extensions, and simulation experiments and applications. Near the bottom are small grain Primitive Utilities and Services, which include data mapping utilities, such as polygonal face triangulation, and CPU-intensive computation services, such as boolean set operations. In the middle are Auxiliary Data Structures that partition surfaces into patches to improve the efficiency in using CPU intensive services, and tag solids to support IC topography design. Near the top are Aggregate Utilities and Services, which perform high level geometrical operations, such as boundary deformation and source visibility. Using Aggregate operations, performance tests and simulation experiments can be written in about 250 to 350 lines of C++ code.

The principal test vehicle for exploring TCAD organizational issues in this thesis is the Berkeley Topography Utilities (BTU) system. The BTU system uses a hierarchical interface approach to integrate existing topography simulators and general-purpose geometry servers. At the bottom, the system integrates SAMPLE-3D, SIMPL, and the IBM Geometry Engine as Centralized Geometry Servers under a hierarchical server interface. When supplemented with Simulation Support Utilities for task management and visualization, it is estimated that the complete BTU system (Centralized Geometry Servers, Hierarchical Server Interface, and Simulation Support Utilities) can be used to implement rigorous simulation applications in about 2,000 lines of C++ code. The BTU system software along with SAMPLE-3D and SIMPL are currently available to the TCAD community. However, BTU performance testing utilities and auxiliary data structures are currently implemented only on the IBM Geometry Engine. A section that follows will discuss how these utilities and data structures can be adapted for use with other boundary representation solid modelers.

Since standardized performance tests take into account the nature of geometrical operations, they are an indispensable system tool for characterizing the run time consequences of theoretical performance bounds. Standardized performance tests can screen out false performance bottlenecks often predicted from simpler asymtotic performance estimates. For example, in the IBM Geometry Engine, computing solid intersection curves between a planar topography and its vertical deposition volume results in a bucket with $O(N)$ triangles, and $O(N^2)$ face bounding box intersection checks. At first glance, for the special case of updating planar topographies, computing solid intersection curves appears to require $O(N^2)$ time. However, performance test results show the run time of solid intersection curve computation still grows as *O(N)* for planar topographies with 100 to 1,000 surface triangles. In fact, the *O(N^2)* face bounding box intersection checks incurred in the planar case cause only 30% run time performance degradation compared to the non-planar case.

Standardized performance tests can also reveal areas where geometry server design tradeoffs interact poorly with IC topography simulation needs. For robustness, most general-purpose boolean set operations create output solids by duplicating geometry components and connectivity links in the input solids. By means of a simple assembly of blocks into a staircase, it is possible to show that, for 1,600 cubes, an IBM Geometry Engine merge operation requires on the average about 10 seconds, regardless of whether the merge is a single block or a set of 40 cubes.

Monotone decomposition is introduced in this thesis as an auxiliary data organization scheme for large grain surface decomposition. In IC topography simulation, large numbers of locally connected facets often have similar orientations. By bin sorting locally connected facets with similar orientations, monotone decomposition can easily partition a simulated surface into large grain monotone patches. Using monotone decomposition, a surface advance with global intra-surface collisions can be broken into a few well-behaved monotone patch advances. Void formations are transformed into overlaps between monotone deformation volumes. In other words, monotone decomposition efficiently focuses the power of merge operations where it is most needed. For example, using monotone decomposition, it is estimated that IBM Geometry Engine merge operations can efficiently resolve void formation in a 1,000 triangle key hole trench surface in about 1 minute.

Finally, this thesis introduces IC topography propagation trace-back as a new TCAD functionality. This functionality allows users to start with a faulty topographical feature, and trace back the process steps and layout masks that might have caused it. Due to the presence of temporary masking layers, such as resist layers, auxiliary data structures are needed to force the propagation of process flow and layout dependencies down to topographical features. These auxiliary data structures can be implemented by semantically extending solid model attribution services.

The key issue in designing semantic extensions is to maintain accurate process flow and layout dependencies, and minimize storage cost of dependencies. This thesis shows that process history tagging, or tagging solid models directly with process step and layout mask id's, is not storage efficient, and limits tagging to only material volumes. As an alternative, this thesis proposes process interaction tracking. For each geometry component, this auxiliary data structure uses a single attribute (e.g. an integer flag), to record the combination of boolean set operations which created the component. Using process interaction tracking, process flow and layout dependencies are compactly represented by a single flag that is automatically updated by each boolean set operation.

9.2. STATUS AND FUTURE DIRECTIONS FOR THE BTU SYSTEM

The BTU system is currently implemented as a BTU directory tree with two major sets of sub-directories. One set of sub-directories are the Server directories that contain Berkeley simulators and wrappers (about 60,000 lines of C code), and IBM servers and simulators (about 50,000 lines of C++ code). The other set of sub-directories are the Interface directories that include standardized performance tests (about 12,000 lines of C++ code), and hierarchical interface utilities (about 41,000 lines of C++ code, including 10,000 lines that duplicate code in standardized performance tests).

Research opportunities exist in proving out the BTU system organization on rigorous physical models. This would require implementing a complete Simulation Support Utilities layer. The easiest way to accomplish this is to wrap SAMPLE-3D's input deck parser and simulation task manager. At present, the SAMPLE-3D wrapper and hierarchical interface utilities already wrap SAMPLE-3D's visualization utilities, such as **pdraw** plot generation, and geometric computation services, such as surface advancement, deloop, and line-of-sight visibility.

The Berkeley part of the Server directories, and the Interface directories are currently available for inspection and use by other TCAD developers. Due to difficulties in obtaining the IBM Geometry Engine, this thesis recommends adapting BTU standardized performance tests and auxiliary data structures to work with commercially available solid modelers, such as ACIS and Echidna. To facilitate this adaptation, the following paragraphs summarize IBM Geometry Engine services and utilities used by the BTU system.

The IBM Geometry Engine provides BTU with a handful of solid modeling services that are commonly found in commercial solid modelers. The IBM Geometry Engine provides BTU with solid geometry primitives, such as spheres and tetrahedra. It also has a piecemeal solid construction interface for adding geometry components one at a time, and invoking the server to establish connectivity links. For standardized performance tests and boundary deformation, BTU uses three IBM Geometry Engine operations: 1) Solid boundary connectivity queries, 2) Point location tests, and 3) Boolean set operations. For topography propagation trace-back, BTU uses IBM Geometry Enginc's attribution mechanism to tag geometry components with name-value pairs. For example, a geometry component that depends on the **Poly 1** mask may be tagged with the ("LayoutDepend", "Poly 1") attribute.

To facilitate application development, the IBM Geometry Engine provides three sets of C++ object classes which are used throughout the BTU system: 1) Memory manager classes, 2) Linked list template classes, and 3) SWR Geometry Server Procedural Interace classes. The first two class sets can be easily re-implemented and re-deployed in the BTU system by a capable C++ programmer in about one person month. The SWR Geometry Server PI classes would require a knowledgeable C++ programmer to replace SWR Geometry Server calls with equivalent operations in the new server.

For use with other solid modelers, the most useful BTU software are the standardized performance tests, the auxiliary data structures, and the simulation experiments. The standardized performance tests can be conducted stand-alone by replacing SWR Geometry Server calls with equivalent new server operations. The auxiliary data structures and simulation experiments require integrating the new server with SAMPLE-3D and SIMPL. This can be accomplished by replacing SWR Geometry Server calls used in hierarchical interface utilities.

9.3. FUTURE RESEARCH IN CENTRALIZING GEOMETRY SERVICES

This thesis explored many fronts in centralizing geometry services. However, there are still many interesting issues and research directions that can be pursued. One interesting research direction is to implement an efficient moving surface geometry server based on the four essential geometry server constructs described in this thesis. This server would be useful both in terms of academic and industrial research. From an academic viewpoint, an efficient 3D moving surface geometry server could be used to uncover other essential constructs. From an industry viewpoint, an efficient 3D moving surface geometry server could immediately become a testbed for deploying physical models for 3D etching and deposition.

It is hoped that the standardized performance tests and auxiliary data structures will be applied to guide the development of other geometry servers, such as ACIS, Echidna, and AT&T's BSP tree solid modeler. The methodology for geometry server performance testing and auxiliary data structure design forwarded by this thesis is generally applicable to boundary representation solid modelers. In spirit, this methodology should perhaps also be applied to the evaluation of 3D field servers, such as PROPHET.

Another research direction is to further study the role of monotone decomposition in computing source and intra-surface visibility. In this thesis, monotone decomposition is shown to be an enabling auxiliary data structure for the real time use of IBM Geometry Engine **merge** operations in 3D boundary deformation. On the other hand, since visibility is a physical phenomenon that simultaneously depends on spatial locality and facet orientation, it seems logical that monotone decomposition may lead to even greater performance improvement in using line-of-sight visibility tests to compute source and intra-surface visibility.

Finally, it would be of interest to implement process interaction tracking for IC topography propagation trace-back, and compare this data structure with small grain process history tagging. Implementing process interaction tracking requires extensive modifications in the geometry server to keep track of different combinations of boolean set operations present in the solid model. As computer memories become cheaper, the improved storage efficiency gained through process interaction tracking may be offset by the complexity in its implementation. Therefore, it would be an interesting experiment to implement process interaction tracking in a geometry server, and compare its accuracy and storage cost to process history tagging on small grain components.