BTU – BERKELEY TOPOGRAPHY UTILITIES FOR LINKING TOPOGRAPHY AND IMPURITY DIFFUSION SIMULATIONS

by

Robert H. Wang

Memorandum No. UCB/ERL M91/71

27 August 1991

BTU – BERKELEY TOPOGRAPHY UTILITIES FOR LINKING TOPOGRAPHY AND IMPURITY DIFFUSION SIMULATIONS

by

Robert H. Wang

Memorandum No. UCB/ERL M91/71

27 August 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

BTU – BERKELEY TOPOGRAPHY UTILITIES FOR LINKING TOPOGRAPHY AND IMPURITY DIFFUSION SIMULATIONS

by

Robert H. Wang

Memorandum No. UCB/ERL M91/71

27 August 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BTU - Berkeley Topography Utilities for Linking Topography and Impurity Diffusion Simulations

Robert H. Wang

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

This report chronicles the development of BTU (Berkeley Topography Utilities) for linking topography and impurity diffusion simulations. Currently implemented on the data structures in the SIMPL-2 program, BTU solves the problem of combining topography and mesh points by providing functions to map topographies between strings generated by topography simulators such as SAMPLE and polygons which can be decomposed into triangular meshes used by impurity diffusion simulators such as SUPREM-IV. To facilitate the integration of topography and impurity diffusion simulators, BTU also includes functions which make a rectangular grid conform to topography, and convert the topography and impurity concentrations between rectangular grid and triangular meshes. The procedural interface is high level in the sense that the functionalities provided by BTU are independent of the underlying SIMPL-2 data structures and geometric algorithms. This allows TCAD developers to add and maintain easily links to other simulators, and gives them the option to reimplement BTU functions for robustness or efficiency as geometric modellers and adaptive grid generators becomes widely available. SIMPL-2 interfaces to SAMPLE and SUPREM-IV are used to demonstrate the procedural interface. Results and run times from SIMPL-IPX simulations of epitaxy with buried layers for submicron twin-well CMOS and BiCMOS processes and a 16-Mb DRAM trench capacitor are presented to demonstrate the simulation capabilities made possible by linking SAMPLE and SUPREM-IV through BTU and measure the performance of the current BTU implementation.

August 18, 1991

In the memory of my father, Mr. Jan-I Wang, who influenced his children in ways he never knew, provided them with opportunities he never had, and loved them in ways they could never repay.

Acknowledgement

First, I would like to thank my research advisor, Professor Andrew R. Neureuther for his support and encouragement throughout this project. Without his vision, insight, and optimistic enthusiasm, this project would never have been realized. I would also like to thank Professor W.G. Oldham for reviewing this report. Thanks also go to Professor R.W. Dutton at Stanford and Professor M.E. Law at University of Florida for useful discussions on SUPREM-IV.

Collaborations with (in alphabetical order by last names) Goodwin Chin at Stanford and Andrej Gabara on SUPREM-IV, Ed Scheckler on SAMPLE and SIMPL-IPX, and Alex Wong on TCAD frameworks are gratefully acknowledged. More than just isolated efforts to finish pieces of an master's project, these experience have helped me grow as a researcher and a programmer.

Personally, these past two years have been some of the more turbulent. I thank God for pulling me through the hard times and blessing me with a wonderful support system full of loving relatives and supportive friends. This report is dedicated to my mother.

The financial support from the Semiconductor Research Corporation, the California State MICRO Program, IBM, and Motorola is gratefully acknowledged.

Table of Contents

List of Figures

1. Introduction

1.1. Motivation

Device scaling into the submicron regime has resulted in complex device topographies which strongly influence impurity profiles and device behavior. To characterize and optimize modern integrated circuit (IC) devices, it is critical to be able to explore accurately and efficiently topography tradeoffs in device design. One cost effective way to perform this task is through integrated process and device simulation. Unfortunately, incorporating topography simulation results into impurity diffusion and device simulations has remained a major obstacle in the development of integrated process and device simulation systems. A key issue which has not been systematically addressed is maintaining the consistency of topography and mesh points at material interfaces.

Topography and mesh points are said to be consistent if they require little modification when they are combined. Ideally, if one uses only data representations which allow efficient implementations of both surface advancement algorithms and adaptive grid generation, such as the octree representation used in Ω [1], topography and mesh points would always be consistent. However, typical topography simulators such as SAMPLE [2,3] and impurity diffusion simulators such as SUPREM-IV [4,5] often use geometrically dissimilar data representations such as strings and meshes. Furthermore, the user may wish to retain different levels of physical details in topography and impurity diffusion simulations. Hence, maintaining the consistency of topography and mesh points is a problem which must be addressed in linking most topography and impurity diffusion simulators.

The problem of maintaining topography and mesh point consistency is especially complex for highly nonplanar structures in which high topography point density is required to represent nonplanar surfaces. As Figure 1.1 illustrates, high topography point density adversely affects the mesh represented using rectangular grids and triangular meshes common in impurity diffusion simulators. In the case of a rectangular grid, combining topography points with the grid requires many grid line additions and results in significant interpolations of impurity concentration. For triangular meshes, there are two strategies for merging topography points into the triangular mesh: merge with remesh and merge without remesh. Merging with remesh results in significant interpolations of impurity concentrations, while merging without remesh may create triangular elements with obtuse interior angles. Figure 1.2 shows why triangular elements with obtuse interior angle are undesirable. As illustrated in Figure 1.2, the impurity diffusion equation is solved numerically using The Divergence Theorem. An obtuse interior angle causes error because it enlarges the area of integration used in the area integral and changes the sign of the normal vectors used in the path integral.

This report chronicles the development of BTU (Berkeley Topography Utilities) for linking topography and impurity diffusion simulations. Currently implemented on the data structures in the SIMPL-2 [6] program, BTU solves the problem of combining topography and mesh points by providing functions to map topographies between strings generated by topography simulators such as SAMPLE and polygons which can be decomposed into triangular meshes used by impurity diffusion simulators such as SUPREM-IV. To facilitate the integration of topography and impurity diffusion simulators, BTU also includes functions which make a rectangular grid conform to topography, and convert the topography and impurity concentrations between rectangular grid and triangular meshes. The procedural interface is high level in the sense that the functionalities provided by BTU are independent of the underlying SIMPL-2 data structures and geometric algorithms. This allows TCAD developers to add and maintain easily links to other simulators, and gives them the option to reimplement BTU functions for robustness or efficiency as geometric modellers and adaptive grid generators becomes widely available. SIMPL-2 interfaces to SAMPLE and SUPREM-IV are used to demonstrate the procedural interface. Results and run times from SIMPL-IPX [7] simulations of epitaxy with buried layers for submicron twin-well CMOS and BiCMOS processes and a 16-Mb DRAM trench capacitor are presented to demonstrate the simulation capabilities made possible by linking SAMPLE and SUPREM-IV through BTU and measure the performance of the current BTU implementation.

1.2. Integration of SAMPLE and SUPREM-IV in SIMPL-IPX

An important goal of this project is to make SIMPL-IPX a viable tool for investigating topography tradeoffs in device design by linking SAMPLE and SUPREM-IV. Another goal is to make the geometric functions developed in this effort reusable for integrating other simulators. To accomplish these goals, a high level procedural interface to access the geometric functions was defined and SIMPL-IPX was reorganized to prove out this interface.

Figure 1.3 shows the current organization of SIMPL-IPX. Prominently displayed in this figure are BTU (Berkeley Topography Utilities) and the simulator interfaces. Figure 1.4 illustrates major components of BTU. At the center of BTU is the geometric modeller which provides functions to traverse and query the data structures. Presently, functions which manipulate the SIMPL-2 polygon and grid data structure form the geometric modeller. One level above the geometric modeller is the geometric toolkit which contains functions to perform computations such as intersection finding and polygon clipping. BTU functions are implemented using a combination of geometric modeller and toolkit functions. Presently, there are seven BTU functions defined in the procedural interface. These functions include **Write_Top**, **Stitch**, **Get_Layers**, **Stitch_Back**, **MC_Grid**, **Mesh**, and **Unmesh**. **Write_Top** converts the top surface of the structure into a string. **Stitch** adds a polygon composed of a deposit string and the top surface. **Get_Layers** converts the structure into a set of strings ordered from top to bottom. **Stitch_Back** clips the structure against an etch string. **MC_Grid** adds grid lines to conform the rectangular grid to the topography. **Mesh** generates a triangular mesh using a topography conforming rectangular grid. **Unmesh** maps impurity concentrations calculated a triangular mesh onto a rectangular grid.

The BTU procedural interface facilitates the linking of additional topography and impurity diffusion simulators by separating geometrical data translation and modification from process modeling. This concept is best illustrated by the SAMPLE and SUPREM-IV interfaces currently in SIMPL-IPX. As shown in Figure 1.5a, interfaces to topography simulators such as SAMPLE can be constructed using the **Write_Top** or **Get_Layers** utilities to create the input strings, and the **Stitch** and **Stitch_Back** utilities to update the structure with the output strings. Similarly, Figure 1.5b shows that interfaces to impurity diffusion simulators such as SUPREM-IV can use the **MC_Grid** and/or **Mesh** utilities to create the input mesh, and the **Unmesh** utility to map the impurity concentrations back onto the topography conforming rectangular grid. In both cases, the TCAD developer only need to generate model information specific to his/her simulator and supply data converters between the simulator data format and either SAMPLE strings or SUPREM-IV triangles.

1.3. Organization

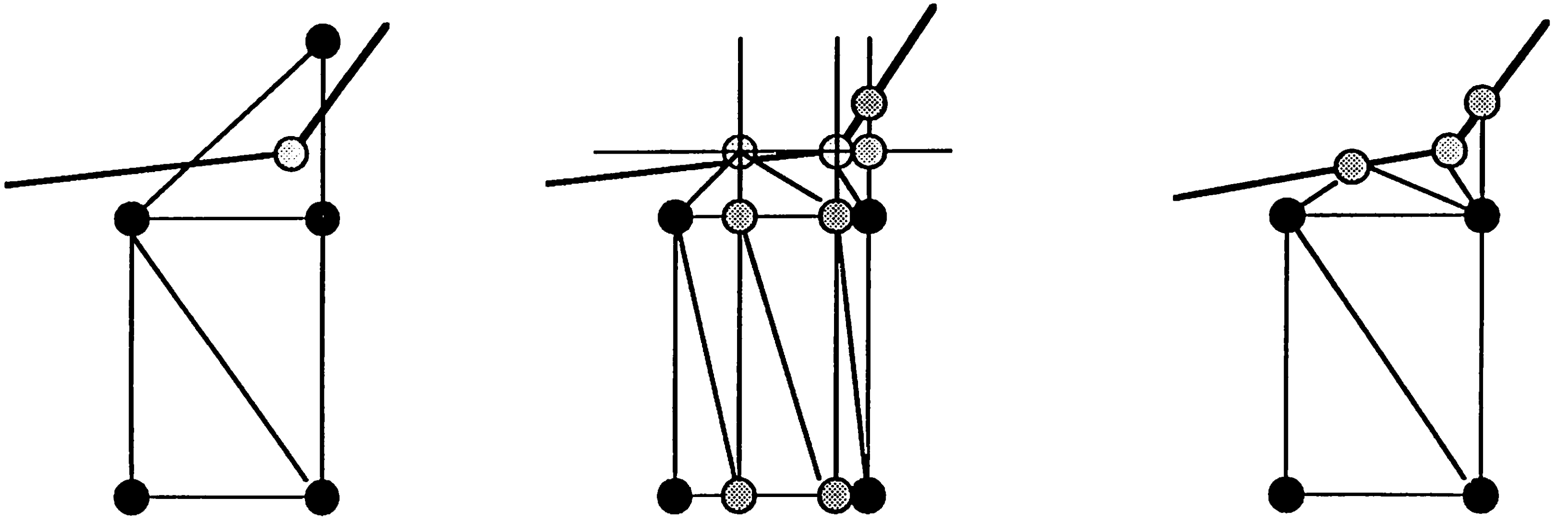
The remainder of the report focuses on the implementation and applications of BTU. Chapter 2 gives a historical account of the development of BTU and contains a detailed discussion of the current implementation. Chapter 3 presents results and run times from SIMPL-IPX simulations of epitaxy with buried layers for submicron twin-well CMOS and BiCMOS processes and a 16-Mb DRAM trench capacitor. Chapter 4 concludes the report with a summary of the contributions of this project and recommendations for future work.

TOPOGRAPHY AND IMPURITY DIFFUSION INTERACTION

HIGH TOPOGRAPHY POINT DENSITY =>

1). SIGNIFICANT GRID POINT ADDITIONS AND INTERPOLATION ERRORS (RECTANGULAR GRID)

2). OBTUSE TRIANGLES (TRIANGULAR MESH)



● = GRID POINTS

◍ = TOPOGRAPHY POINTS

RECTANGULAR GRID

8 POINTS ADDED
0 OBTUSE ANGLES

TRIANGULAR MESH

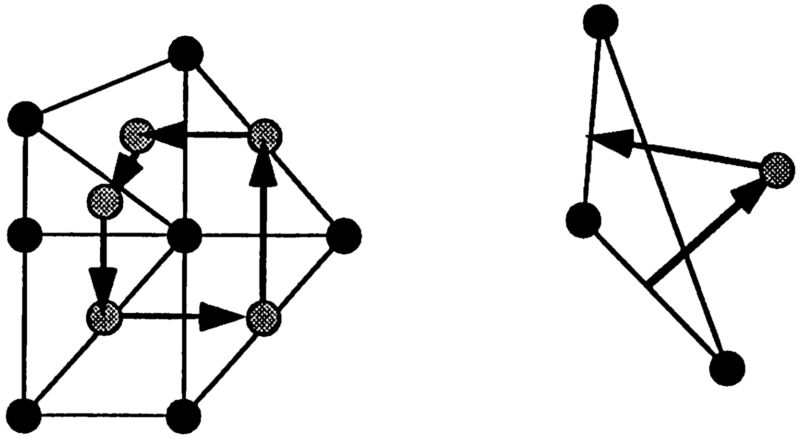
3 POINTS ADDED
3 OBTUSE ANGLES

Figure 1.1

IMPURITY DIFFUSION EQUATION

$$\int_A \text{div}\, J \, dA = \int_s J \bullet \vec{n} \; ds$$

$$\int_A \frac{\partial C}{\partial t} - K_r \left(C_I C_V - C_I^* C_V^*\right) dA = \int_s D \left(\text{grad}\, C + \frac{q}{kT} u_n \,\text{grad}\, V\right) \bullet \vec{n} \; ds$$



OBTUSE INTERIOR ANGLE:

- **ENLARGED AREA OF INTETEGRATION**
- **NEGATIVE NORMAL VECTORS**

Figure 1.2

SIMPL-IPX
SYSTEM ORGANIZATION

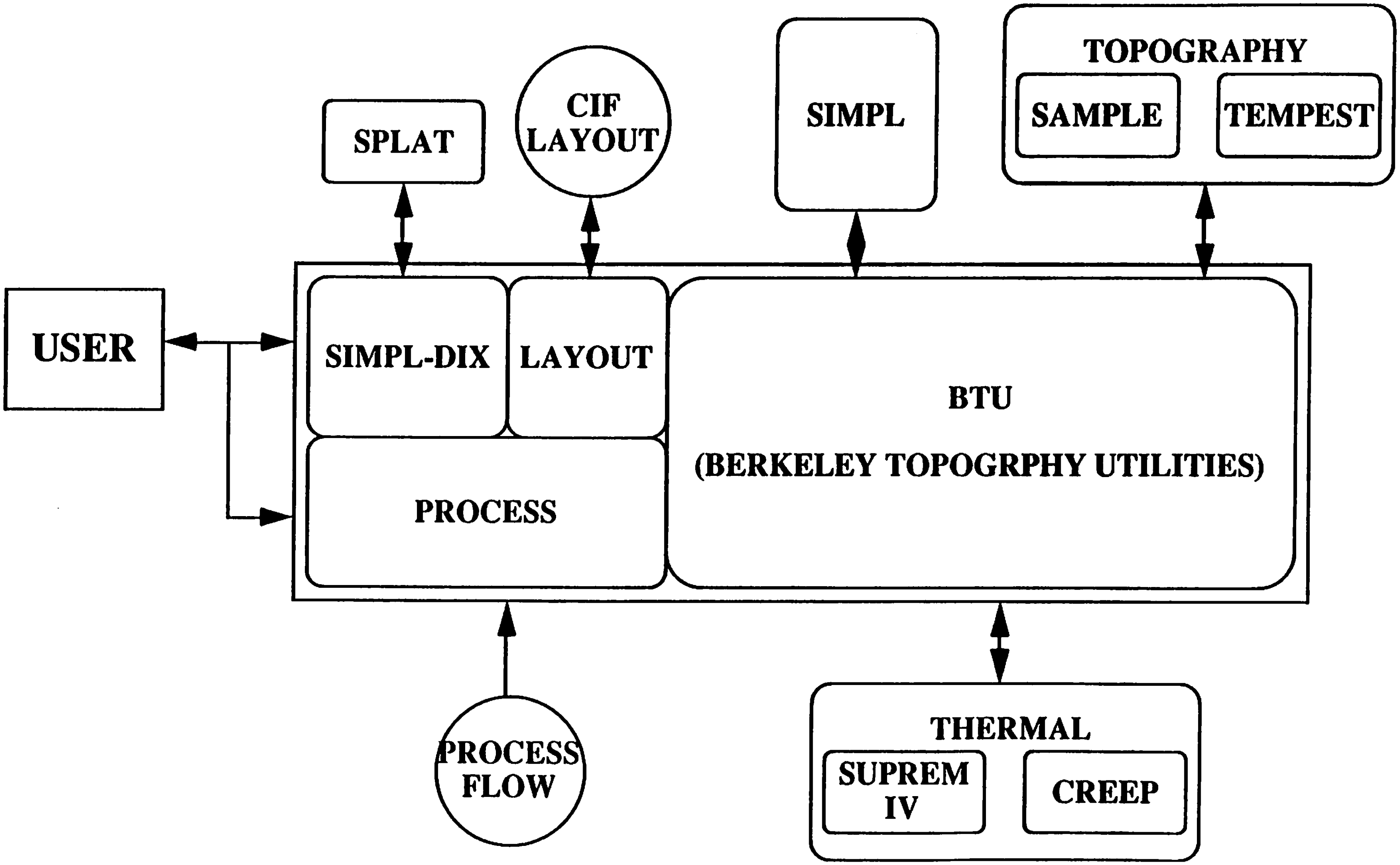
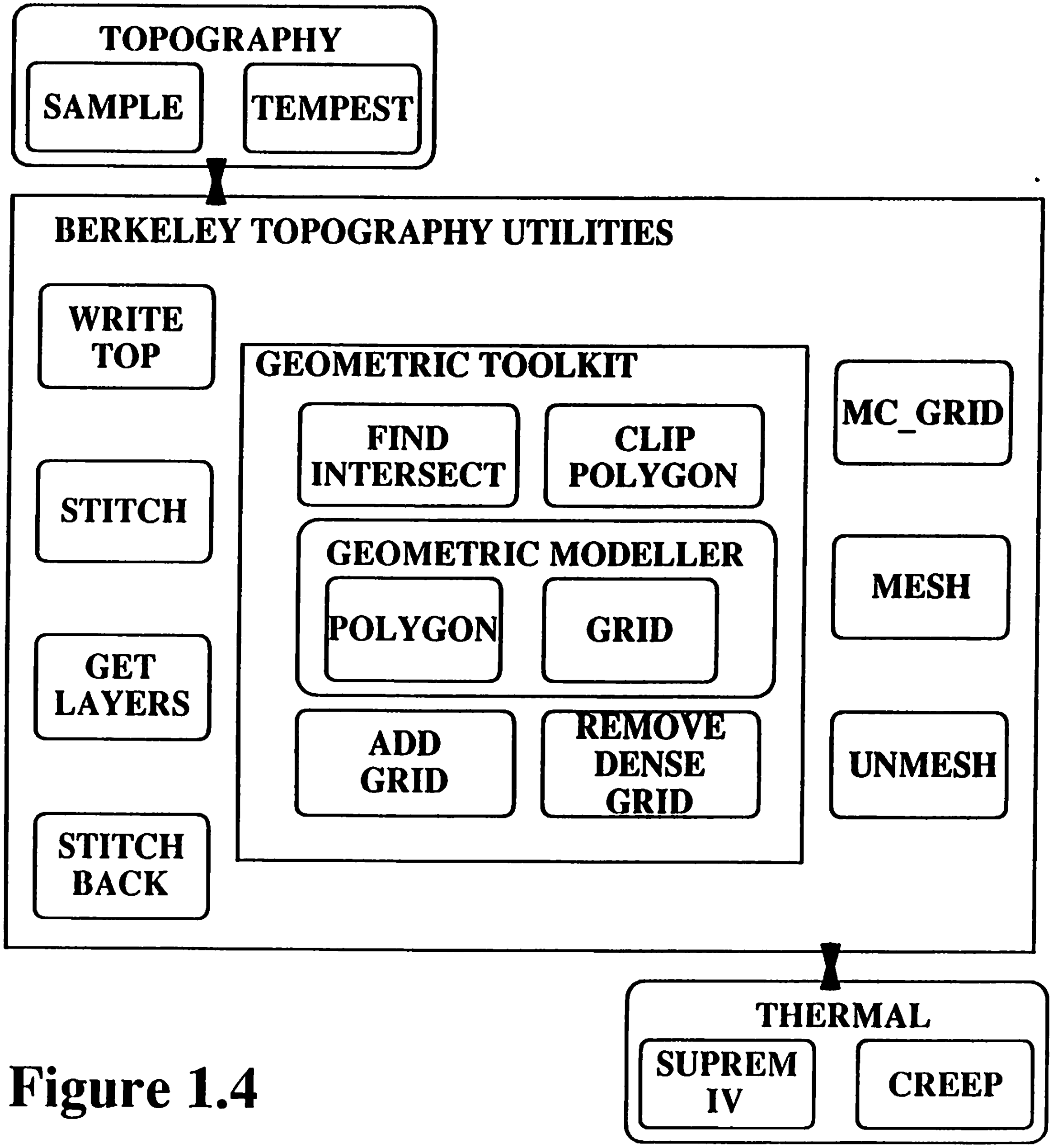


Figure 1.3

BERKELEY TOPOGRAPHY UTILITIES



TOPOGRAPHY

SAMPLE

TEMPEST

BERKELEY TOPOGRAPHY UTILITIES

WRITE TOP

STITCH

GET LAYERS

STITCH BACK

GEOMETRIC TOOLKIT

FIND INTERSECT

CLIP POLYGON

GEOMETRIC MODELLER

POLYGON

GRID

ADD GRID

REMOVE DENSE GRID

MC_GRID

MESH

UNMESH

THERMAL

SUPREM IV

CREEP

Figure 1.4

TOPOGRAPHY SIMULATION INTERFACE

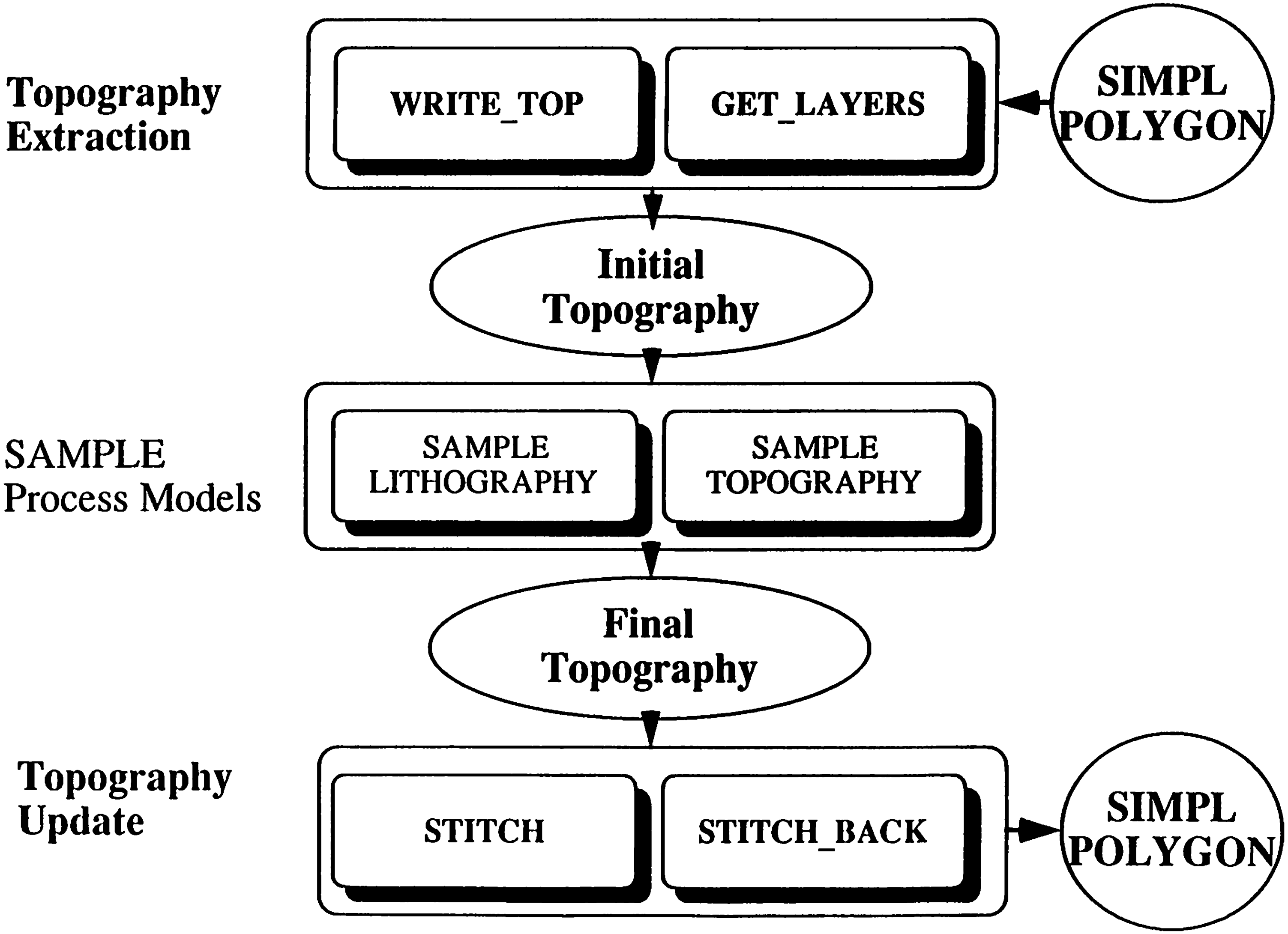
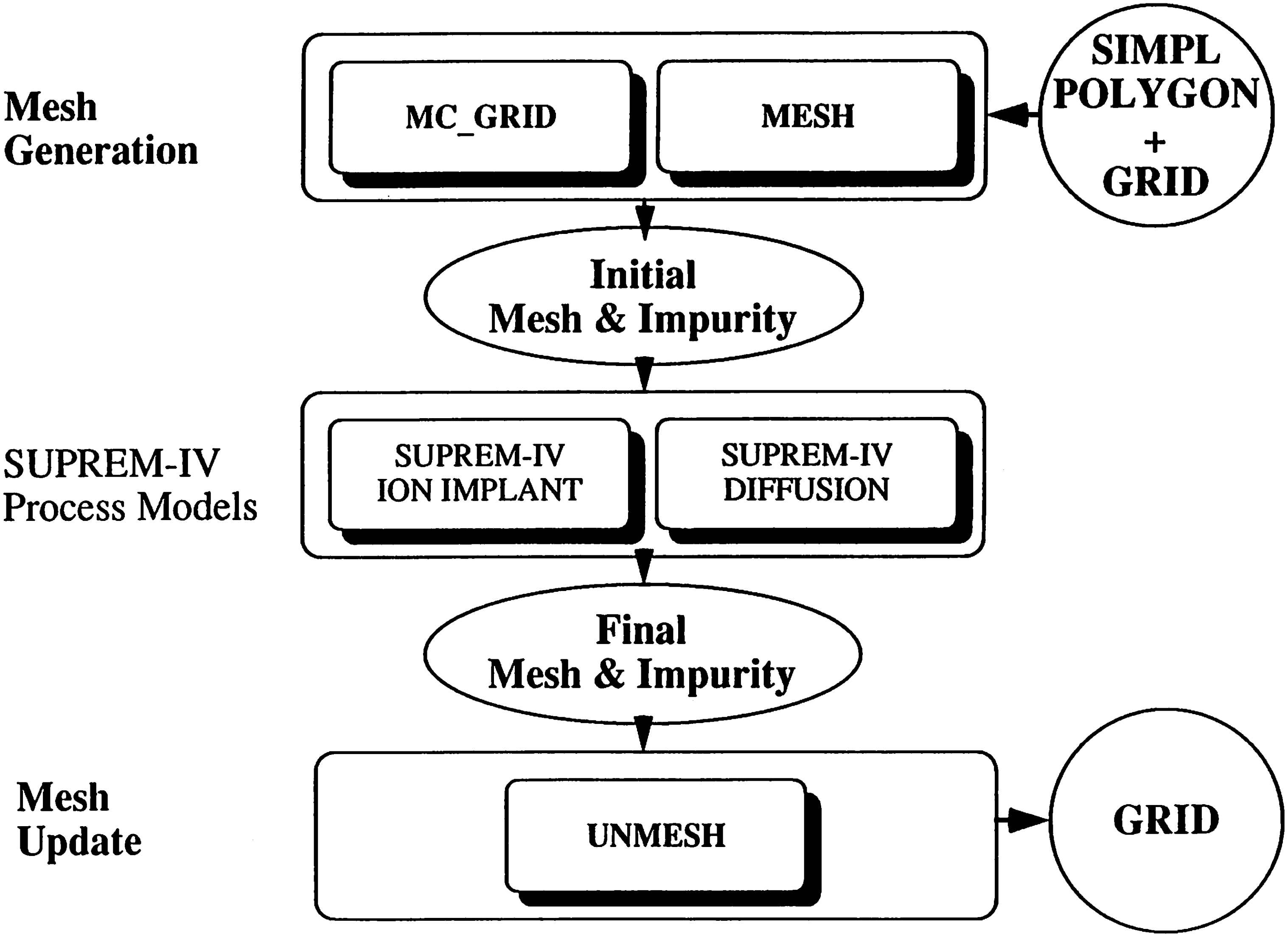


Figure 1.5a

IMPURITY DIFFUSION SIMULATION INTERFACE



Mesh Generation

MC_GRID

MESH

SIMPL POLYGON + GRID

Initial Mesh & Impurity

SUPREM-IV Process Models

SUPREM-IV ION IMPLANT

SUPREM-IV DIFFUSION

Final Mesh & Impurity

Mesh Update

UNMESH

GRID

Figure 1.5b

2. Background, Data Structures, and Algorithms

2.1. Background

The origin of BTU can be traced back to the seminal work by Lee on the SIMPL-2 program. Lee introduced a polygon data structure for topography simulation and a rectangular grid data structure for impurity diffusion and device simulations. The geometric functions which he implemented to traverse and query these data structures lay the foundation for the geometric modeller currently used by BTU. Lee also implemented the first BTU algorithms, the **Write_Top** and **Stitch** utilities, as part of the interface for SAMPLE deposition.

The next milestone in the development of BTU algorithms was the implementation of the **Get_Layers** utility by Scheckler for etching and parasitic extraction of VLSI interconnects [8]. To avoid the problem of clipping the structure against the etch string, Scheckler simulated etching only on structures covered by blanket deposition and updated these structures by deleting the top polygon and adding the etch string to the structure using the **Stitch** utility. This "strip and stitch" approach was also used later by Wang and Scheckler to update the structure after lithography simulations [7].

The "strip and stitch" approach was, however, not adequate for simulating etching of highly nonplanar structures such as the silicon trench. Furthermore, it was also recognized that complete characterization of advanced IC device structures such as the silicon trench required linking topography and impurity diffusion simulations. Thus, work began in Fall 1989 to extend the SAMPLE interface for general topography simulation and construct a SUPREM-IV interface for ion implant and impurity diffusion simulations. By Spring 1990, collaborative efforts between Wang at the University of California at Berkeley working on SIMPL-2 and

Chin at Stanford University working on SUPREM-IV resulted in SIMPL-IPX and successfully demonstrated integrated topography and impurity diffusion simulations for a bipolar process with self-aligned polysilicon emitter and polysilicon collector plug [7]. However, most of the utility algorithms were embedded in the SAMPLE interface in SIMPL-2 and SUPREM-IV process modules and required rather circular operations to invoke. Consequently, the prototype SIMPL-IPX lacked a clear structure to support modular growth of additional utility algorithms or interfaces to other topography and impurity diffusion simulators.

The current definition and implementation of BTU came as the result of reorganizing SIMPL-IPX to support modular growth. All BTU algorithms were either reimplemented or developed during the reorganization. The **Write_Top**, **Stitch**, and **Get_Layers** utilities were taken out of the SAMPLE interface and reimplemented as independent BTU functions. The **Stitch_Back** utility which formerly relied on a pseudo plasma etching module in SUPREM-IV was redesigned to work with utilities which operate directly on the SIMPL-2 rectangular data structure. A **MC_Grid** utility was created to make rectangular grids conform to topography. **Mesh** and **Unmesh** utilities were created to transfer impurity concentrations between rectangular grids and triangular meshes.

2.2. Data Structures

The current implementation of BTU algorithms uses the SIMPL-2 polygon and grid data structures which are illustrated in Figure 2.0. The polygon data structure is composed of a linked list of polygons, each containing a loop of vertices and a material name, and a network of vertices. Each vertex in the network contains the coordinates of the vertex, pointers to vertices that follow it in the network, and the material names attached to the edges corresponding to the vertex pointers. The material names attached to an edge is that of the polygon on the right side of the edge. Common polygon data structure operation include **Get_Left_Top**, **Get_Right_Top**, and **Move**. **Get_Left_Top** finds the vertex which 1) is exposed to air, 2) is on the simulation window boundary, and 3) has an "x" value equal to "xmin". **Get_Right_Top** performs a similar task except it finds the vertex that has an "x" value equal to "xmax". **Move** takes as input a vertex and a material name, and returns the vertex at the end of the edge that corresponds to the material name.

The grid data structure is made up of two one-dimensional arrays which store the locations of the vertical and horizontal grid lines, and a three dimensional array which stores the impurity concentrations. The first index of the three dimensional array indicates the type of impurity, while the second and third indices of the three dimensional array point to the coordinates of the grid point. In the current version of SIMPL-2, only boron and arsenic concentrations are supported. Operations to insert and delete grid lines and rows and columns of impurity concentrations are available. However, these operations are computationally expensive, i.e. O(n^2) where n is number of grid lines, because both the grid line and impurity concentrations arrays are static.

2.3. Algorithms

There are currently seven functions defined in the BTU procedural interface: **Write_Top**, **Stitch**, **Get_Layers**, **Stitch_Back**, **MC_Grid**, **Mesh**, and **Unmesh**. The sections below describe the functionalities of the utilities and the algorithms used in the current implementation.

2.3.1. Write_Top

The **Write_Top** utility converts the top surface of the structure to a string. First, **Write_Top** calls the **Get_Left_Top** and **Get_Right_Top** utilities to get the left and right end points of the top string. Then, starting at the left end point, all the vertices which are exposed to air are traversed until the right end point is reached. Figure 2.1 lists the pseudocode and illustrates the Write_Top utility algorithm. A more detailed description is given in [6].

2.3.2. Stitch

The **Stitch** utility calls the **Write_Top** utility to get the top surface of the structure and then adds a polygon composed of a deposit string and the top surface. The new polygon is created by connecting the end points of the deposit string and the top surface. The pseudocode and a pictorial description of the **Stitch** utility is given in Figure 2.2. A more detailed description can also be found in [6].

2.3.3. Get_Layers

The **Get_Layers** utility converts the structure into a set of strings ordered from top to bottom by successively deleting polygons and calling the **Write_Top** utility. The most important component of the **Get_Layers** utility algorithm is the **Find_Top_Polygon** function, which determines if a polygon sits on top of all the other polygons in the structure. A detailed description of the criteria used to determine the "top polygon" is in [8]. Figure 2.3 outlines and illustrates the effects of the **Get_Layers** algorithm.

2.3.4. Stitch_Back

The **Stitch_Back** utility clips the structure against an etch string by finding and inserting into the structure intersections between the polygons and the etch string, stitching together the intersections and string points into a linked list of vertices, and clipping the polygons in the structure using the etch string represented by the linked list of intersection and strings. Figure 2.4a gives the pseudocode which outlines the major steps in the **Stitch_Back** operation.

To ensure that CPU time is not wasted sifting through collinear or relatively collinear points during the **Stitch_Back** operation, a geometric toolkit function, **Work_Out_Top**, is usually invoked before **Stitch_Back** to filter the etch string. As shown in Figure 2.4b, **Work_Out_Top** throws out string points which join segments that are extremely short or have equal slopes.

The task of finding and inserting intersections is performed by the geometric toolkit function **Find_Intersect** illustrated in Figure 2.4c. As shown in Figure 2.4c, **Find_Intersect** is a brute force, O(Nn) algorithm which compares N polygon segments against n string segments. Intersections computed by **Find_Intersect** are inserted into the structure and sorted by their relative distances to string points for use in the stitching stage. In the current implementation of **Stitch_Back**, most of the CPU time is spent on **Find_Intersect** since it is an O(Nn) operation. By comparison, stitching the intersections and string points is an O(n) operation and the **Clip_Polygon** algorithm is O(N). However, by calling **Work_Out_Top** before **Stitch_Back** with a tolerance of 1.0e-06, most etch strings can be reduced down to below 400 points, and can be processed by Stitch_Back in about 10 CPU seconds on an IBM RS/6000 Model 530.

The computation of intersections is actually done using the geometric modeller function **Two_D_Search_or_Insert2**. Using a test which checks if a point lies counter-clockwise, collinear, or clockwise to a segment, **Two_D_Search_or_Insert2** determines if an intersection exists between a polygon segment and a string segment by testing if 1) the polygon vertices lie on opposite sides of the string segment and 2) the string points lie on opposite sides of the polygon segment [9]. Cases covered by this criterion are illustrated in Figure 2.4d. If an intersection does exist, **Two_D_Search_or_Insert2** calculates the coordinates of the intersection by parametrizing the polygon and string line segments using the standard $y = mx + b$ form and solving the resultant system of line segment equations. The intersection is inserted into the structure using the geometric modeller function **Insert**, which properly establishes connectivity between the intersection and vertices originally in the structure.

The geometric toolkit function **Clip_Polygon** is used to cut polygons with an etch string represented as a linked list of intersections and string points. For each polygon, **Clip_Polygon** first determines if the polygon should be deleted, clipped, or kept in tact. For polygons which are clipped, **Clip_Polygon** first identifies all the cuts into the polygon made by the etch string and then traverse these cuts and some of the original polygon vertices to create one or more polygons. Figure 2.4e illustrates how Clip_Polygon uses cuts to create new polygons. Experience with **Stitch_Back** has shown the robustness of **Stitch_Back** depend heavily on the robustness of **Clip_Polygon.** The current version of **Clip_Polygon** works well for cutting structures such as lines and trenches and updating etchback away from material interfaces, but is less successful at updating etchback close to material interfaces.

2.3.5. MC_Grid

MC_Grid, which is short for Make_Conform_Grid, defines the topography on the rectangular grid by finding and inserting into the structure intersections between the polygons and the grid lines, adding grid lines at all polygon vertices, and removing grid lines which are surrounded by tiny grid spacings. **MC_Grid** is the utility responsible for combining polygon vertices and mesh points. Figure 2.5a gives a pseudocode listing which highlights the major steps in **MC_Grid** and illustrates its effect on the rectangular grid data structure.

MC_Grid calls the geometric toolkit functions **Find_x_Intersect** and **Find_z_Intersect** to find vertical and horizontal grid line intersections and insert these intersections into the structure. As shown in Figure 2.5b, the functionalities of **Find_x_Intersect** and **Find_z_Intersect** are similar to that of **Find_Intersect**, except in this case polygon segments are compared against grid lines rather than string segments.

Once grid line intersections have been included the structure, the geometric toolkit function **Add_Grid**, illustrated in 2.5c, is invoked to drop grid lines on every vertex in the structure. In essence, this step maps the topography onto the rectangular grid.

Clearly, adding grid lines on every vertex in the structure introduces more mesh points than what most impurity diffusion simulators can handle. For instance, a typical SAMPLE string of 250 points roughly results in the addition of 250x250 or 62,500 mesh points which an order of magnitude larger than the size limit of the SUPREM-IV mesh point table of 6000 mesh points. Hence, after **Add_Grid**, **MC_Grid** invokes the geometric toolkit function **Remove_Dense_Grid** to sort all grid line pairs by their spacings and then remove grid lines which are surrounded by tiny spacings. After the initial round of grid line removals, if the number of mesh points still exceeds the mesh point table size, spacing tolerance is increased to remove more grid lines. This process continues until the number of mesh points is less than the size limit of the mesh point table.

2.3.6. Mesh

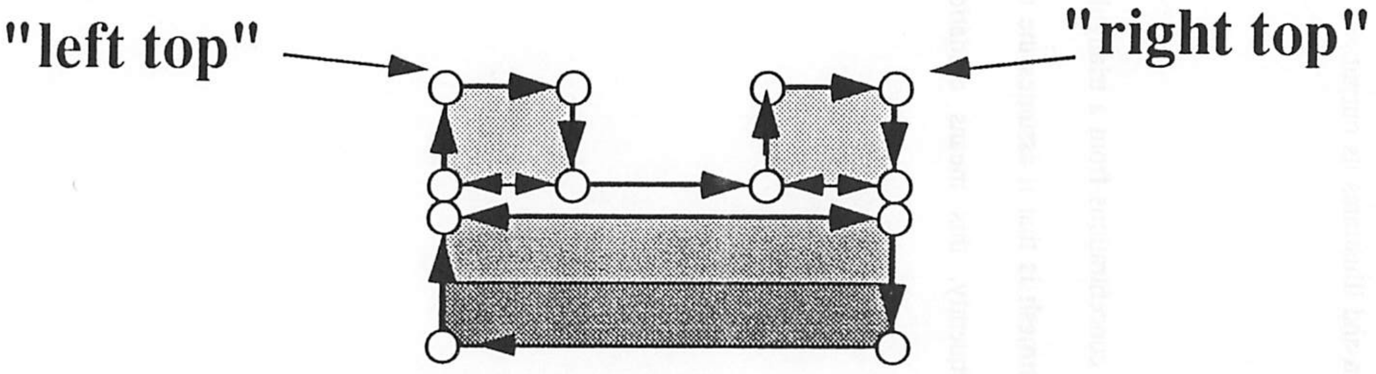
The **Mesh** utility generates a triangular mesh from a topography conforming rectangular grid. The assumption that the topography is defined at grid points greatly simplifies the algorithm since it implies that string segments would lie on the diagonals of the rectangular element. Figure 2.6 outlines the **Mesh** algorithm and illustrates its output.

2.3.7. Unmesh

The **Unmesh** utility maps the impurity concentrations from a triangular mesh back to the rectangular grid. The major limitation of **Unmesh** is that it assumes the topography does not change during impurity diffusion. Consequently, this means oxidation during impurity diffusion is not currently supported by BTU.

SIMPL-2 DATA STRUCTURES

- POLYGONS



- RECTANGULAR GRID

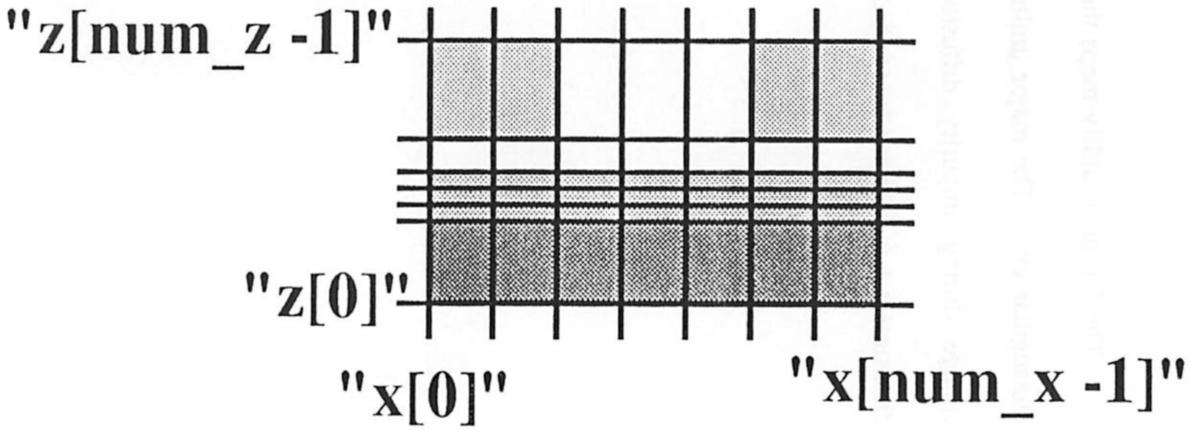


Figure 2.0

WRITE_TOP

Write the top surface

```
Write_Top
{
   /* Data Conversion */
  Find "left top" vertex;
  Find "right top" vertex;
 vertex = "left top" vertex;
 n_top = 0;
 while (vertex != "right top" vertex) {
     top[n_top] = vertex;
     n_top++;
     vertex = next vertex that contains air;
  }
}
```

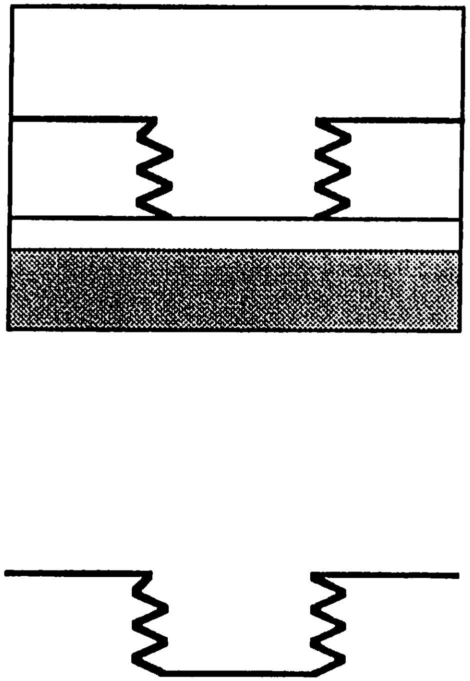


Figure 2.1

STITCH

Add deposit string to top surface

```
Stitch
{
  /* Input Data Filter */
   top = Work_Out_Top;

  /* Data Conversion */
  bottom = Write_Top;
  Create polygon using top and
    bottom layer;
  Insert_Polygon;
 }
```

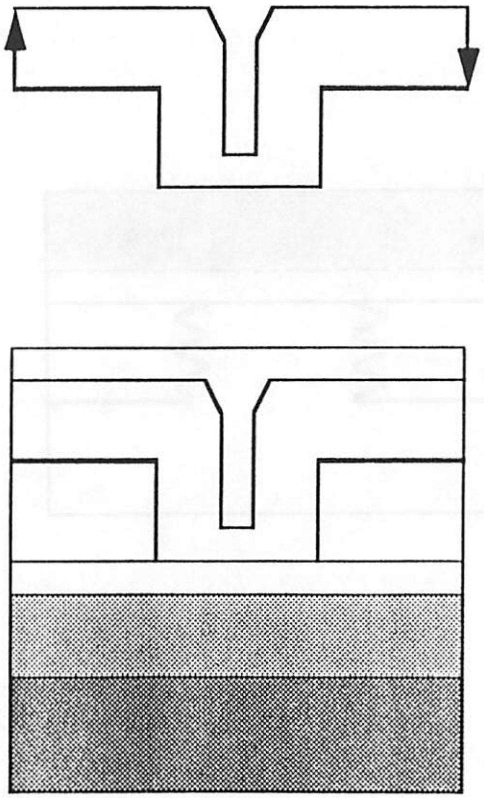


Figure 2.2

GET_LAYERS

Convert polygons to strings

```
Get_Layers
{
   Save cross section;

  /* Data Conversion */
  n_layers = 0;
  while (more polygons left) {
     layers[n_layers] = Write_Top;
     n_layers++;
     Find "top" polygon;
     Delete "top" polygon;
  }

   Restore cross section;

  /* Output Data Filter */
  Remove redundant layers;
}
```

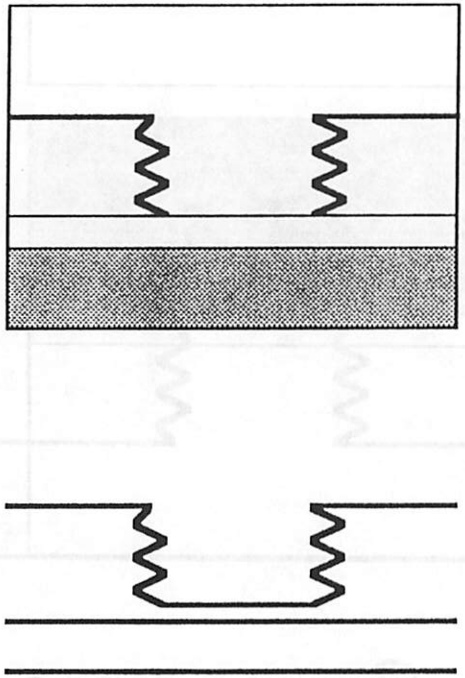


Figure 2.3

STITCH_BACK

Clip polygons against etch string

```
Stitch_Back
{
   /* Input Data Filter */
    Set_EtchWin();

   /* Data Conversion */
    Find_Intersect();
    Stitch_EtchFront();
    Mark_Polygon();
    foreach (polygon) {
       Clip_Polygon(polygon);
    }
  }
```

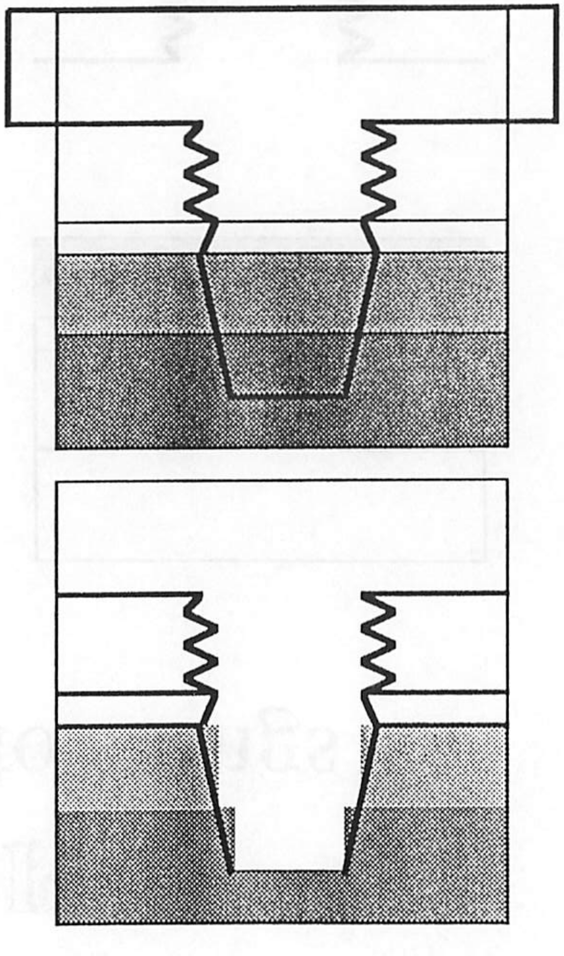
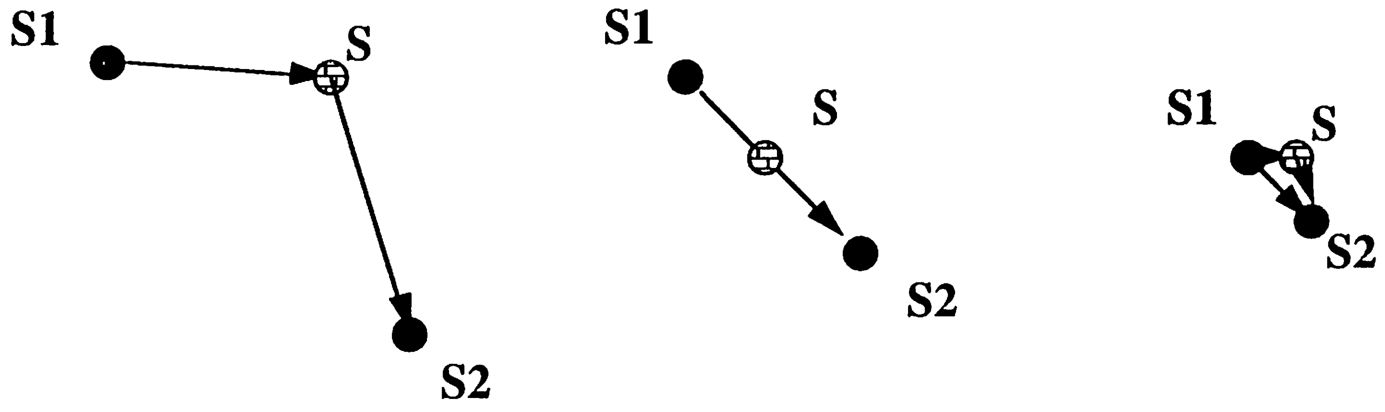


Figure 2.4a

STITCH_BACK

Work_Out_Top

Filter collinear or dense string points



(S1,S) and (S,S2) Have Different Slope and are Long
Keep S

(S1,S) and (S,S2) has Same Slope
Delete S

(S1,S) and (S,S2) are Very Short
Delete S

Figure 2.4b

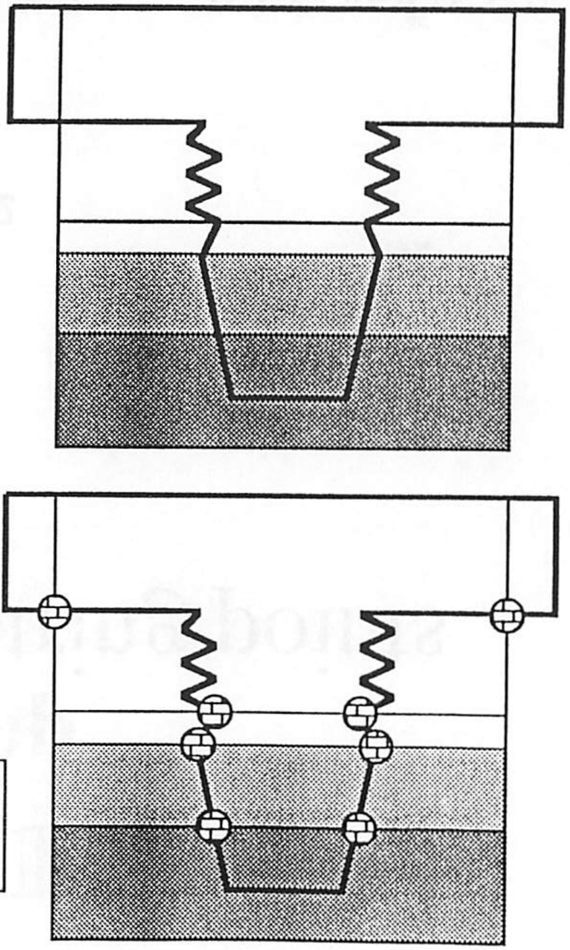
STITCH_BACK

Find_Intersect

Find polygon and string intersections

```
Find_Intersect
{
   foreach (polygon segment) {
      foreach (string segment i) {
           /* Compute and insert intersection
              into cross section */
           Two_D_Search_or_Insert2(V_int, ...);

           /* Insert intersections into a
              sorted link list of intersections */
           Insert_Etch(IntList[i], V_int, ...);
      }
   }
}
```



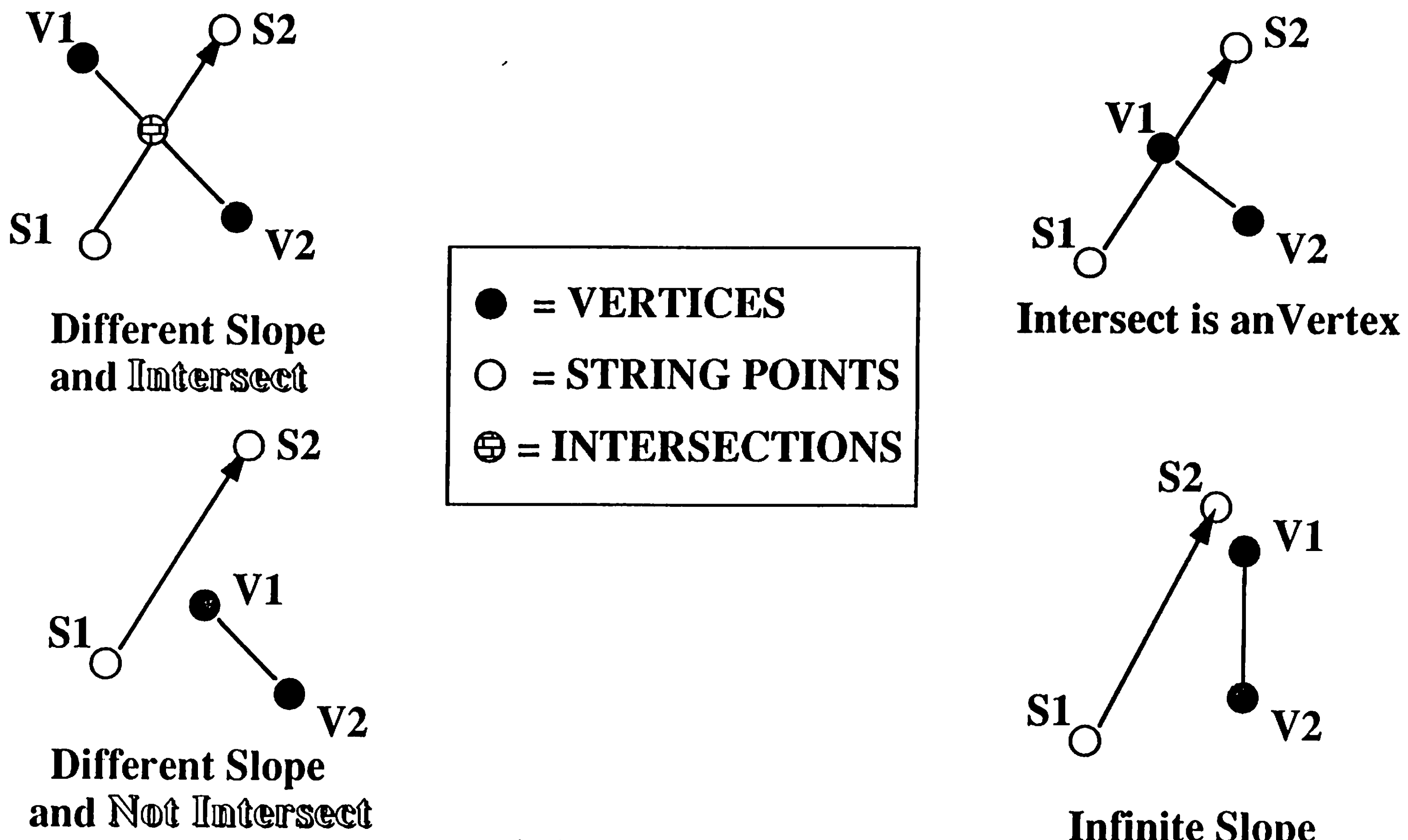
⊕ = INTERSECTIONS

Figure 2.4c

STITCH_BACK

Two_D_Search_or_Insert2

Find polygon and string segment intersections



Different Slope and Intersect

Intersect is anVertex

Different Slope and Not Intersect

Infinite Slope

Figure 2.4d

STITCH_BACK

Clip_Polygon

Clip polygon against cuts

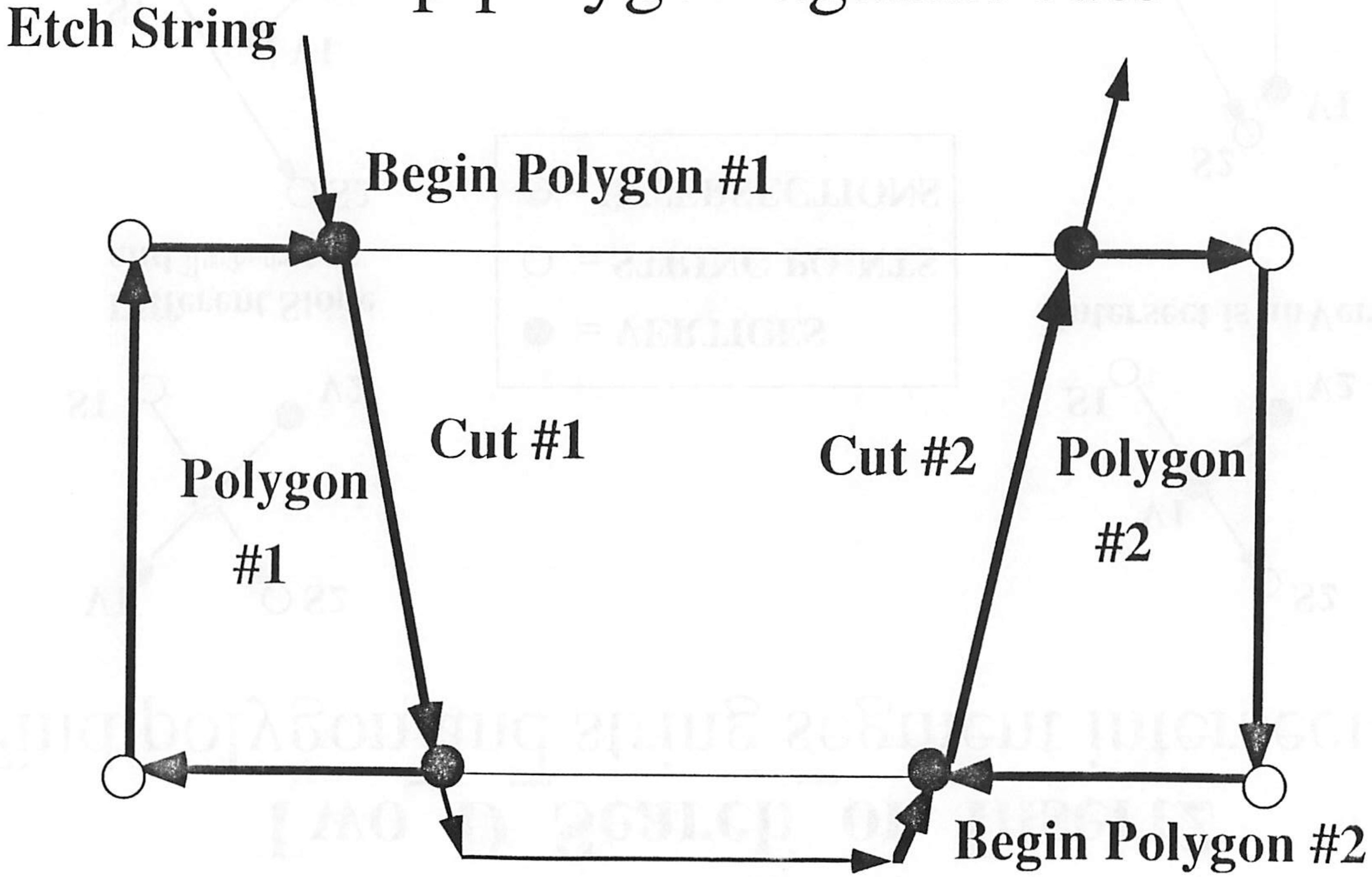


Figure 2.4e

MC_GRID

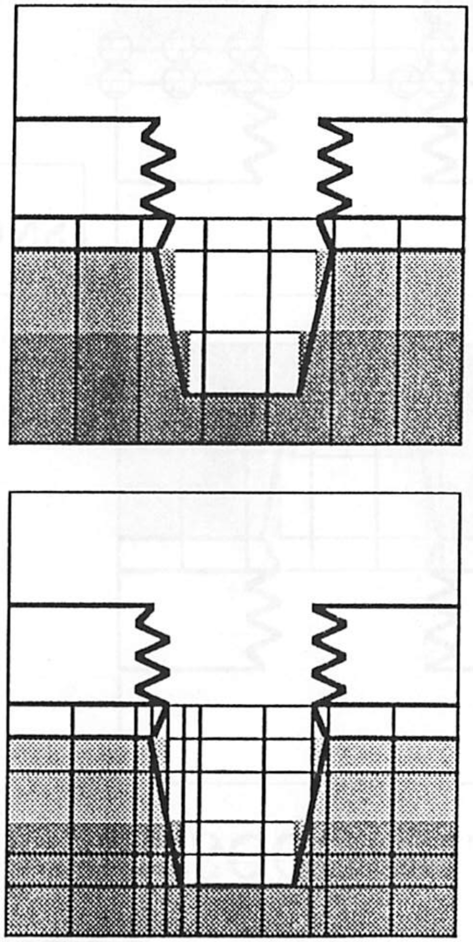
Make rectangular grid conform to the topography

```
MC_Grid
{
  /* Input Data Filter */
  Save cross section;
  Delete_Row();

  /* Data Conversion */
  Find_x_Intersect();
  Find_z_intersect();

  /* Data Conversion */
  Add_Grid();

  /* Output Data Filter */
  Remove_Dense_Column();
  Remove_Dense_Row();
  Restore cross section;
}
```



NOTE: Curved SideWalls

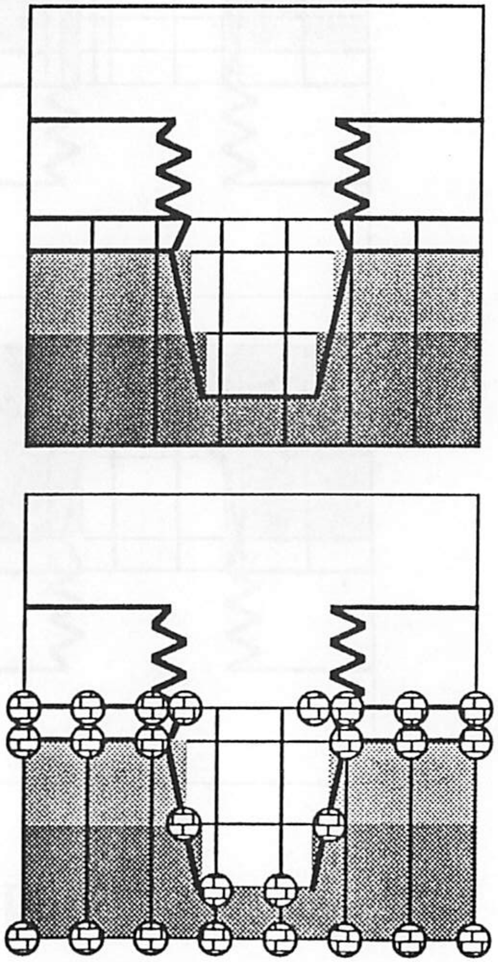
Figure 2.5a

MC_GRID

Find_Grid_Intersect

Find polygon and grid line intersections

```
Find_x_Intersect
{
   foreach (polygon segment) {
      foreach (vertical gridline i) {
          /* Compute and insert intersection
             into cross section */
          Two_D_Search_or_Insert2(V_int, ...);
      }
   }
}
Find_z_Intersect
{
   foreach (polygon segment) {
      foreach (horizontal gridline i) {
          /* Compute and insert intersection
             into cross section */
          Two_D_Search_or_Insert2(V_int, ...);
      }
   }
}
```



⊕ = INTERSECTIONS

NOTE: Curved Side Walls

Figure 2.5b

MC_GRID
Add_Grid
Drop grid lines on vertices

Grid Line Intersection (x,z):

Do Nothing;

Horizontal Grid Line Intersection (x,z) :

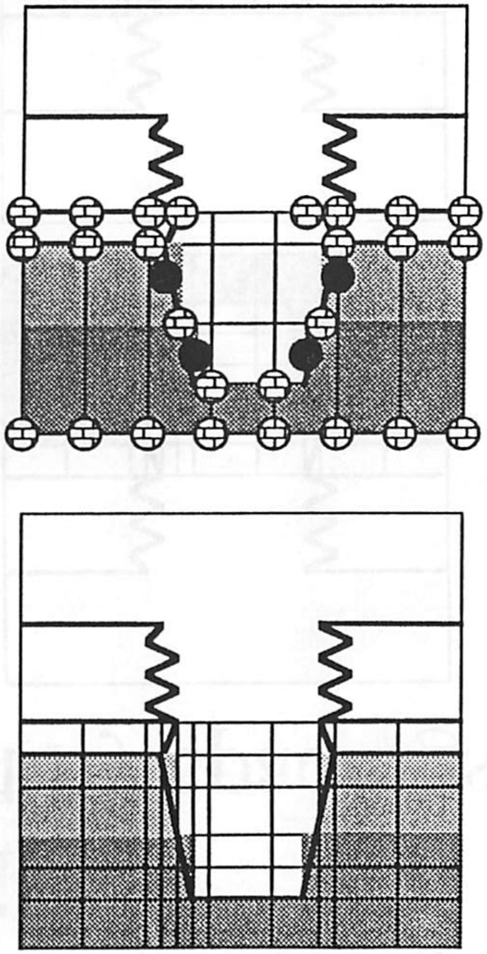
Add Vertical Grid Line at x;
Interpolate doping along x;

Vertical Grid Line Intersection (x,z) :

Add Horizontal Grid Line at z;
Interpolate doping along z;

Vertex (x,z) :

Add Vertical Grid Line at x;
Interpolate doping along x;
Add Horizontal Grid Line at z;
Interpolate doping along z;



● = VERTICES

⊕ = INTERSECTIONS

Figure 2.5c

Assume Curved Side Wall

MC_GRID

Remove_Dense_Grid

Delete grid lines surrounded by tiny spacings

```
Remove_Dense_Grid
{
   Sort each  grid line pair by spacing;

   Determine critical spacing such that after
     removal :  n_grid_lines < max_n_grid_line;

   foreach (grid line i) {
     if (grid line i is in two critical grid line pairs) {
         Remove_Grid_Line(i);
     }
   }
}
```

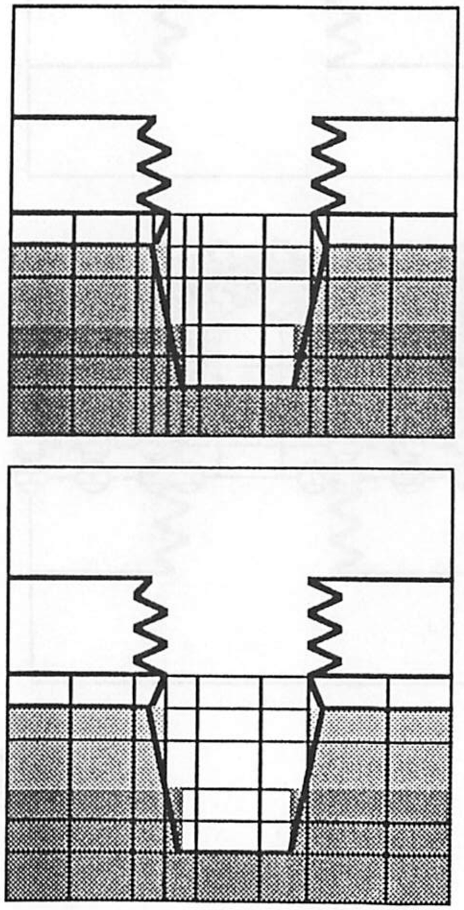


Figure 2.5d

MESH

Generate triangular mesh from rectangular grid

```
Mesh
{
  /* Input Data Filter:
     Generate valid grid points */
  Read cross section;
  foreach (xz pair)
    if ((x,z) is in cross section)
      Write grid point;

  /* Data Conversion:
     Generate triangles */
  foreach (rectangular element)
    if (element is in bulk)
      Write both triangles;
    else if (element is at surface)
      Write conforming triangle;
}
```

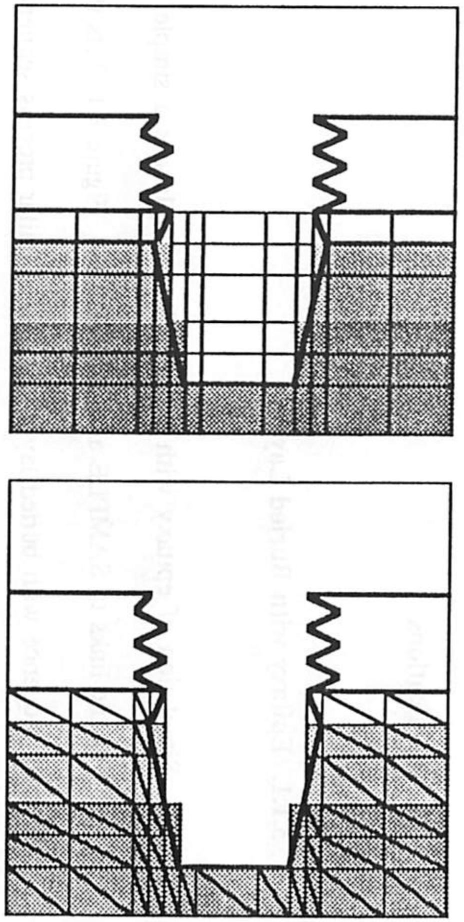


Figure 2.6

3. Applications and Run Time Performance

3.1. Applications

3.1.1. Epitaxy with Buried Layers

Simulation of epitaxy with a buried layer implant is a simple yet practical example for verifying the links to SAMPLE and SUPREM-IV. Figure 3.1a lists the key steps in an epitaxy process sequence with buried layer implants. Similar process sequences are used used in sub-micron twin-well CMOS and BiCMOS processes [10]. Processing begins with a 580 A oxide growth which is followed by an 800 A pad nitride deposition. A window is opened at the center of the mask for an arsenic buried N+ layer implant of 1.0e+15 cm^{-2} and 80 keV. A diffusion step of 25 minutes and 1100 C is used to drive-in the buried N+ layer implant. The oxide growth during the diffusion step is simulated analytically using SIMPL-2. Using the field oxide as a mask, a boron buried P layer implant of 1.0e+15 cm^{-2} and 80 keV is introduced. After an N epitaxial growth of 1.3 um and 1.0e+15 cm^{-3}, a diffusion step of 5 minutes and 1100 C is used to activate the buried P layer implant and simulate autodoping.

Figure 3.1b shows the cross section after the activation of buried N+ layer. Results from the buried P layer implant are shown in Figure 3.1c while Figure 3.1d plots the corresponding SUPREM-IV mesh. The cross sections before and after the autodoping simulation are shown Figures 3.1e and 3.1g respectively. Figure 3.1f shows the SUPREM-IV mesh tuned to simulate autodoping. For epitaxy and film deposition, several horizontal grid lines must be added to the grid generated by MC_Grid to capture doping transitions in the deposited material. Meshes generated by only BTU functions would be devoid of mesh points in the bulk of the deposited material because **Stitch** does not place any polygon vertices along the edges of the deposited polygon and **MC_Grid** places grid lines only on polygon vertices.

EPITAXY with BURIED LAYERS

- N+ BURIED LAYER IMPLANT
- N+ BURIED LAYER DRIVE
- **FIELD OXIDE GROWTH**
- P BURIED LAYER IMPLANT
- EPITAXIAL GROWTH
- AUTODOPING

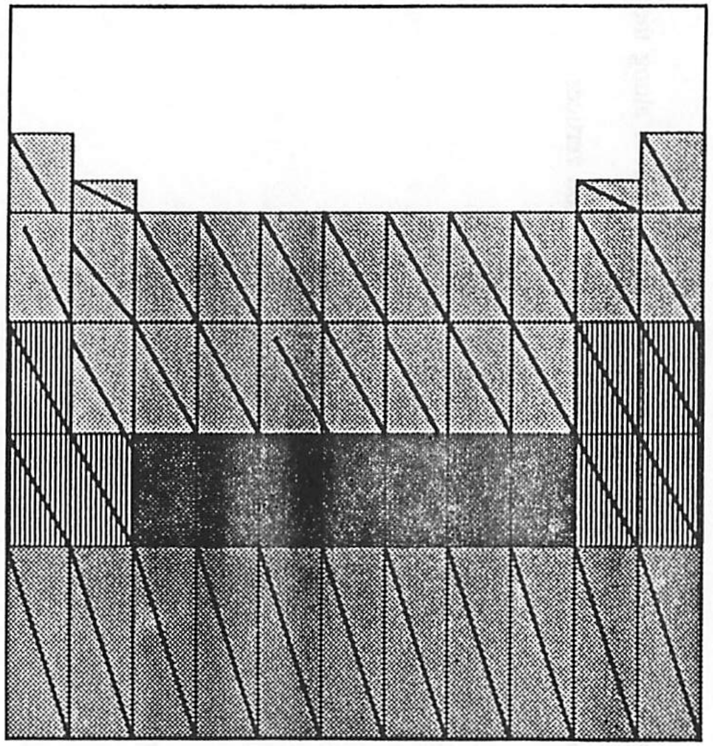
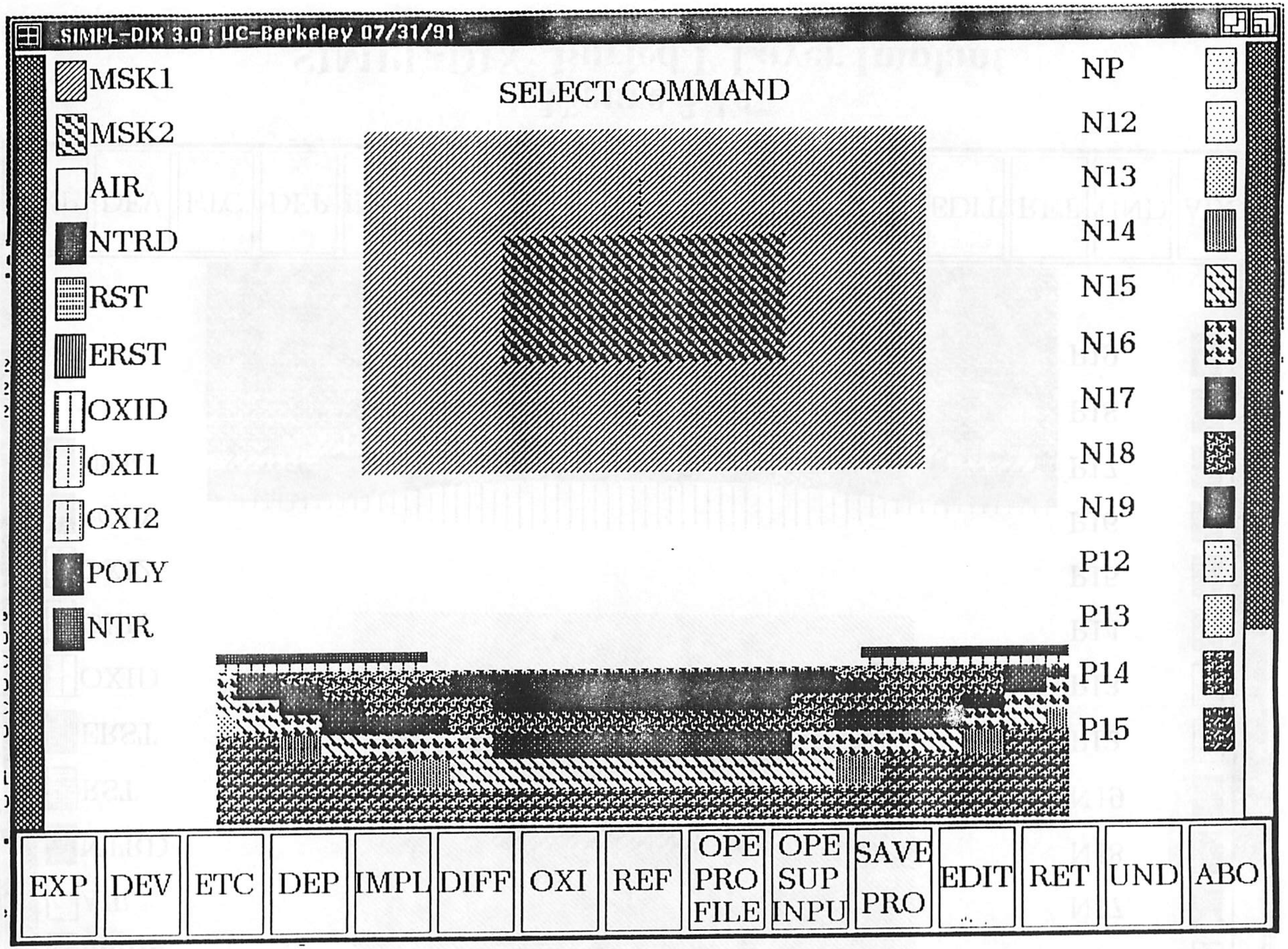
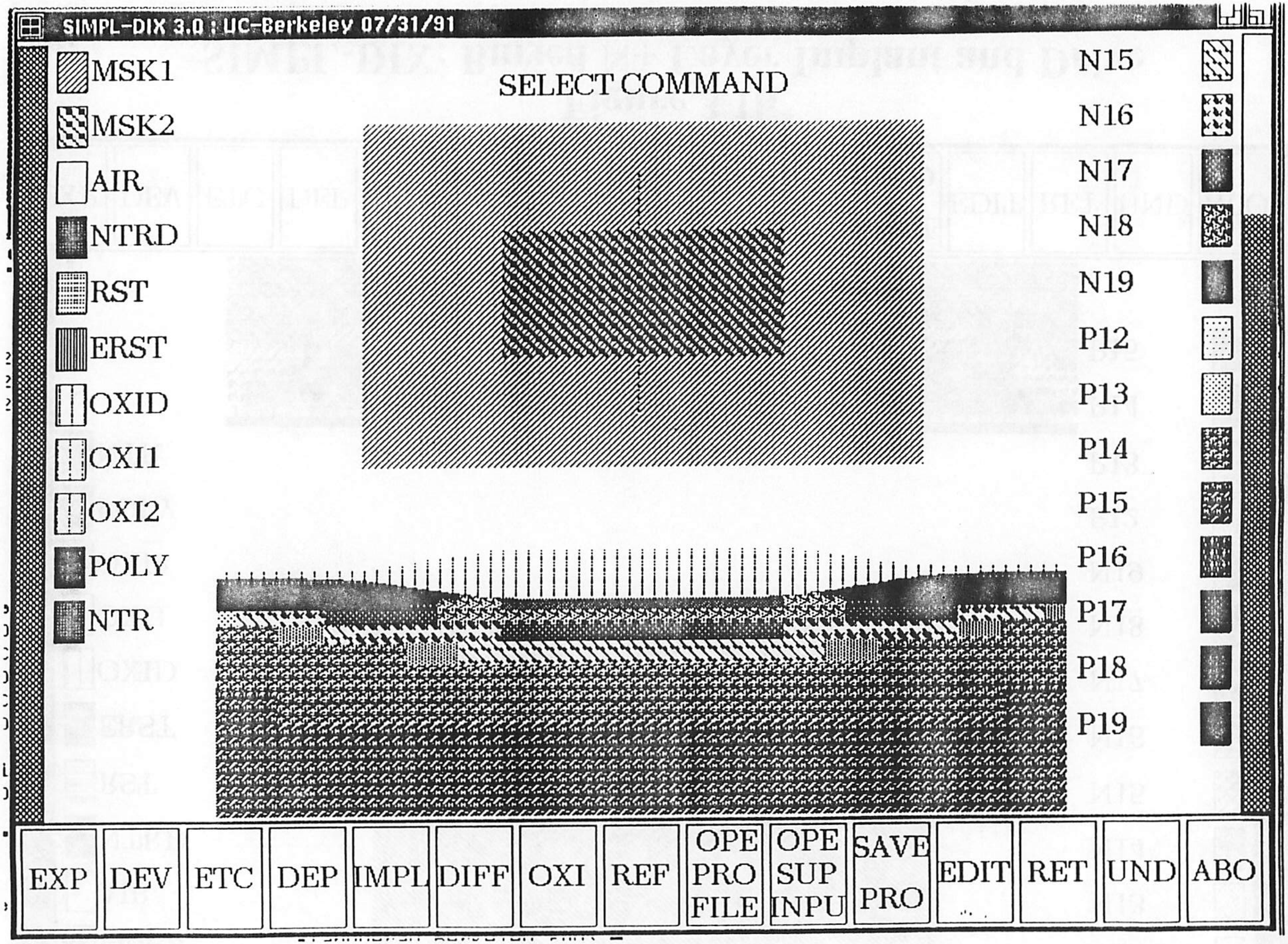


Figure 3.1a


SIMPL-DIX 3.0 : UC-Berkeley 07/31/91

SELECT COMMAND

Figure 3.1b
SIMPL-DIX: Buried N+ Layer Implant and Drive



SIMPL-DIX 3.0 ; UC-Berkeley 07/31/91

SELECT COMMAND

Figure 3.1c
SIMPL-DIX: Buried P Layer Implant

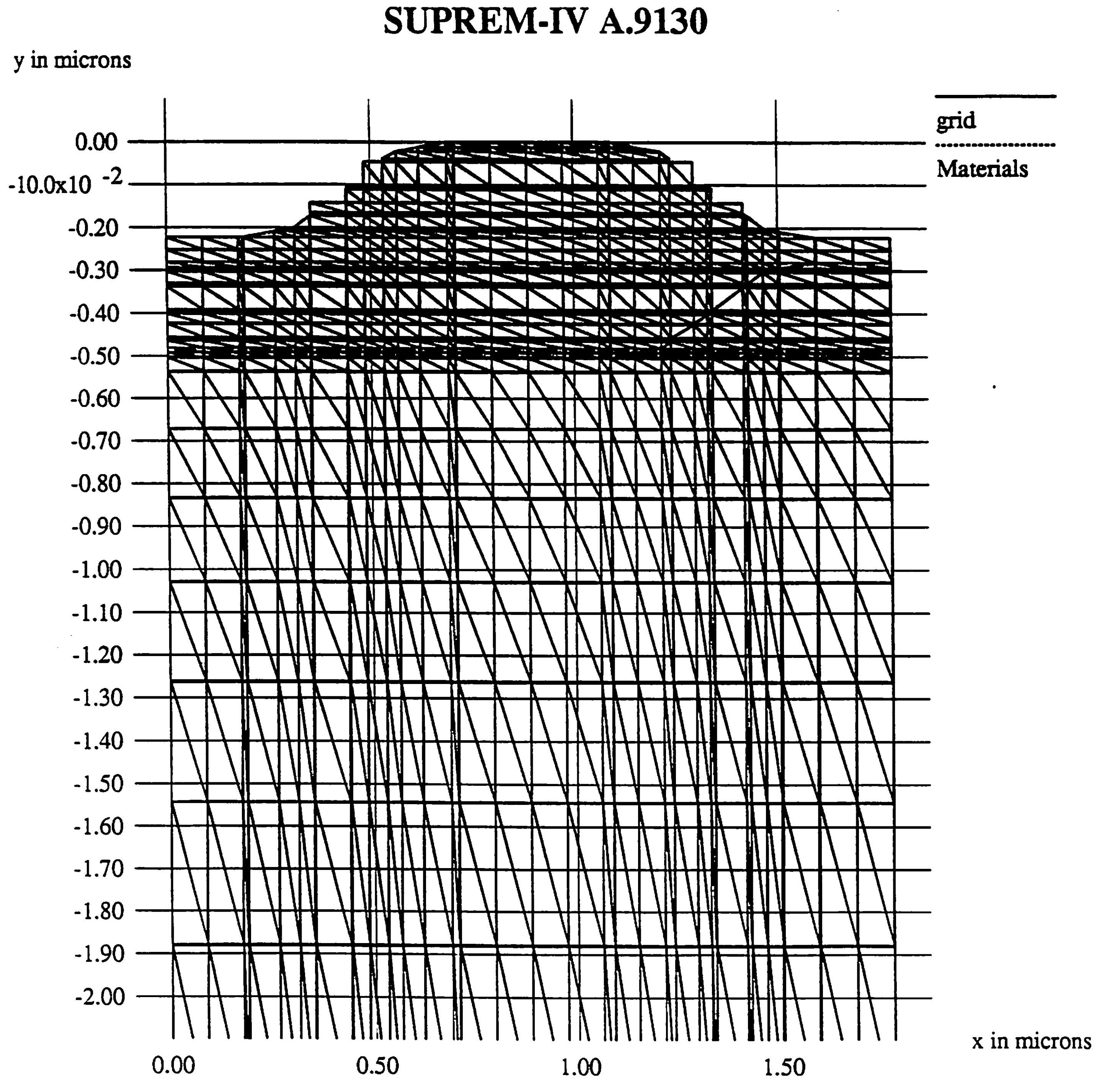


Figure 3.1d
SUPREM-IV Mesh: Buried P Layer Implant

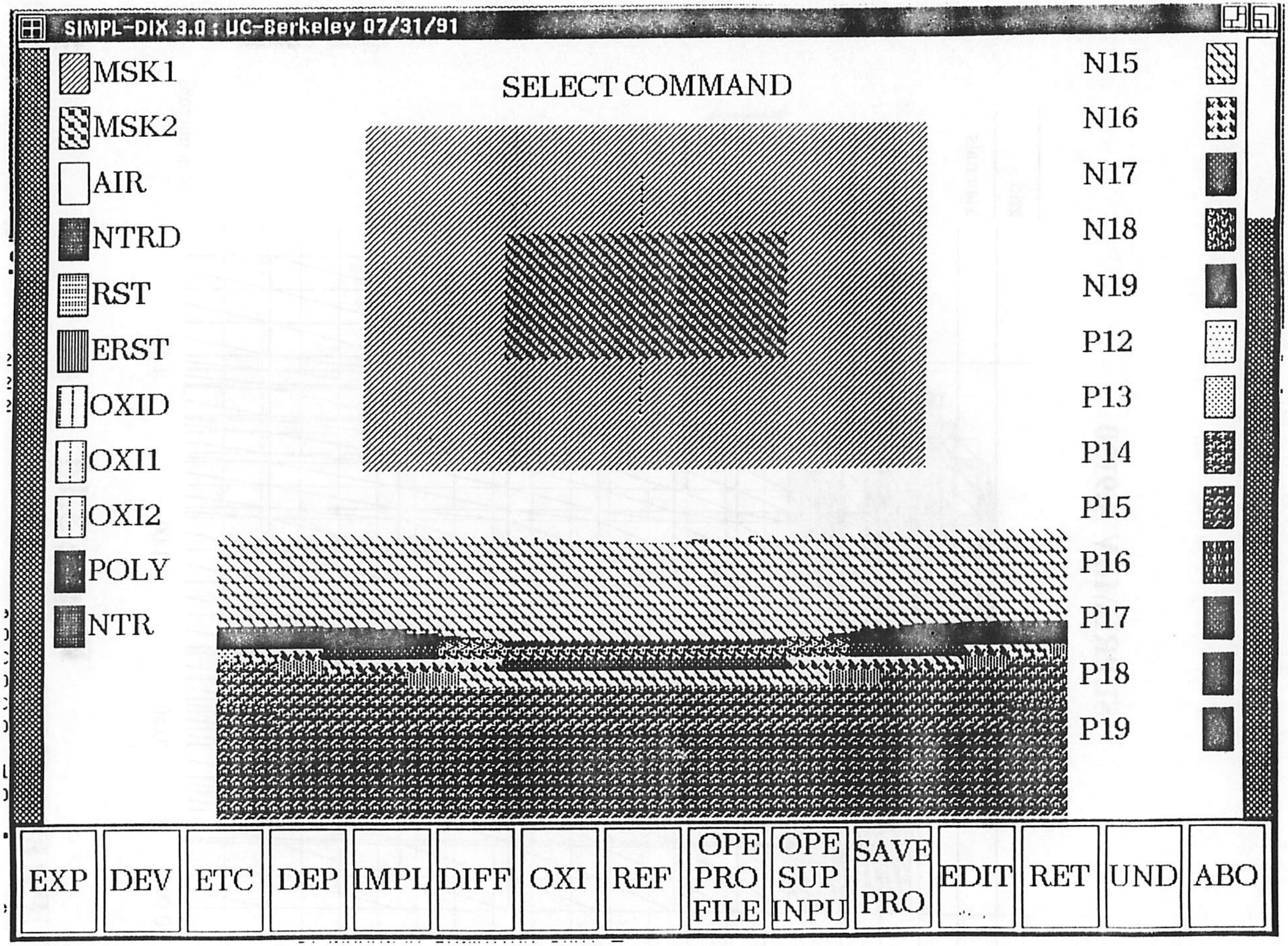


Figure 3.1e
SIMPL-DIX: N Epitaxy

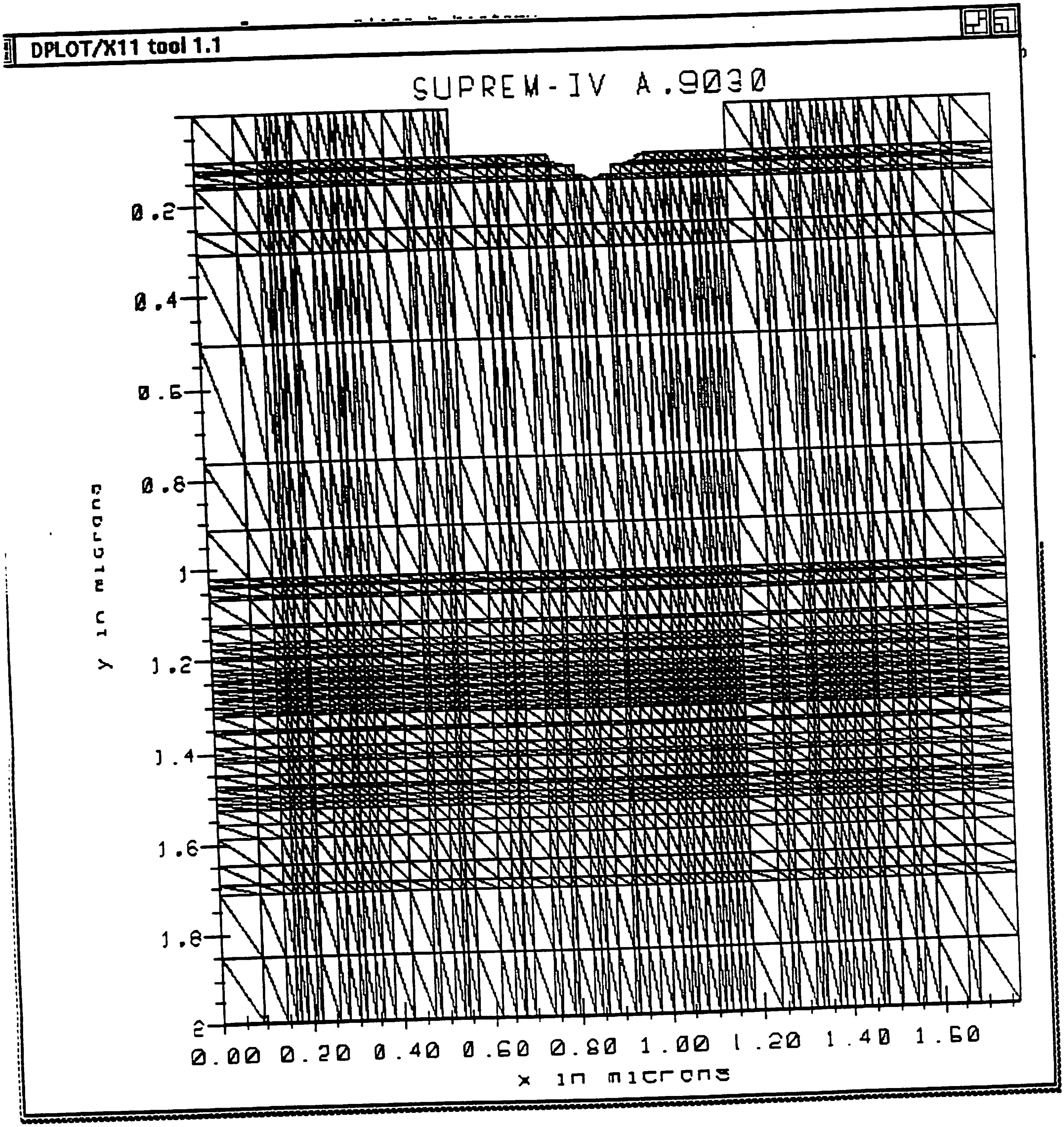
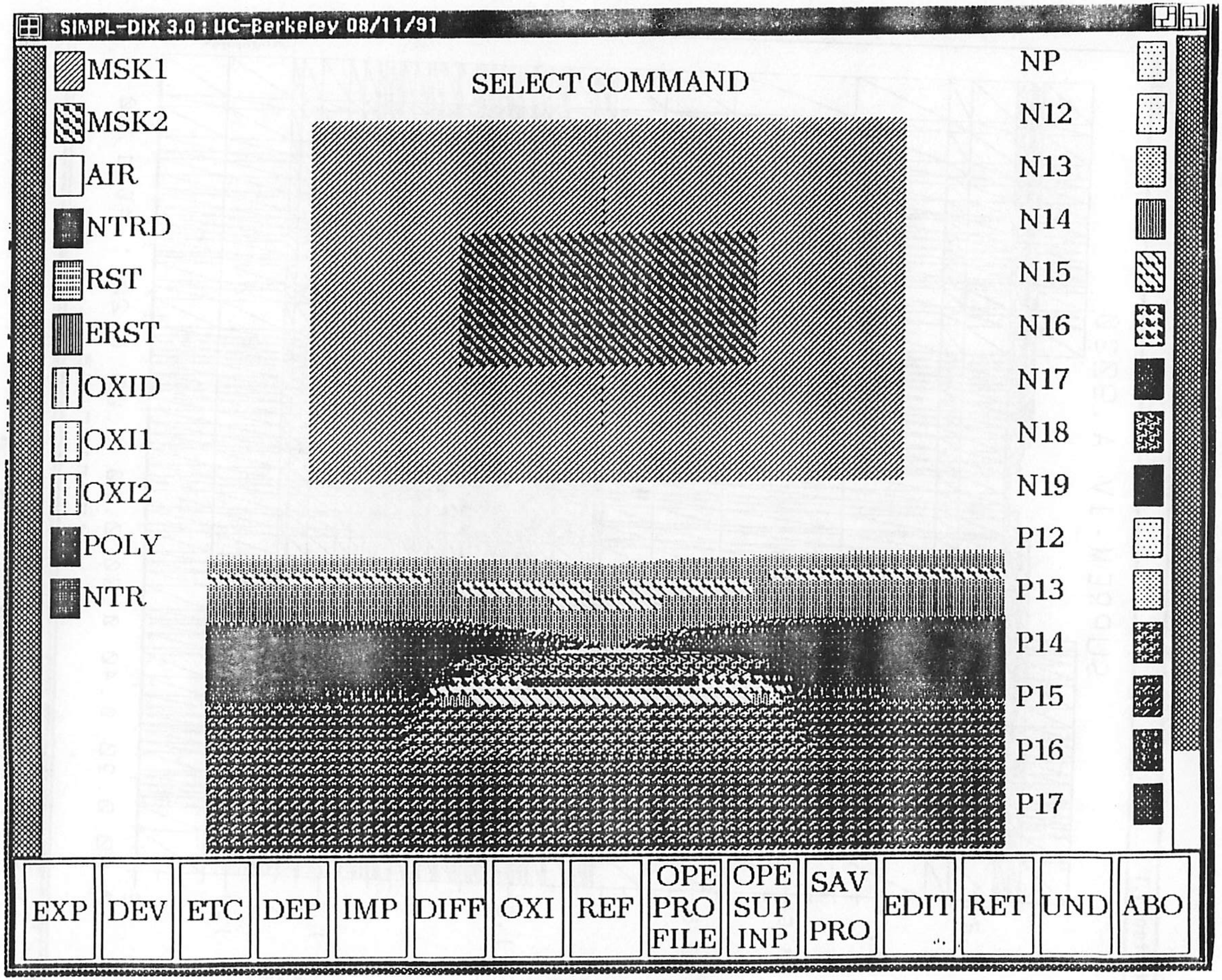


Figure 3.1f
SUPREM-IV Mesh: Autodoping



SELECT COMMAND

Figure 3.1g
SIMPL-DIX: Epitaxy with Buried Layers

3.1.2. 16-Mb DRAM Trench Process

Trench process simulation is an excellent performance benchmark for BTU functions due to the topographical complexity and the strong topography and impurity profile interaction inherent in trench structures. Figure 3.2a outlines the major steps of a 16-Mb DRAM trench process similar to the one described in [11]. First, several layers of materials are vertically deposited using SIMPL-2. These layers include an P epitaxial layer of 2 um and 1.0e+13 cm^{-3}, and a 5000 A oxide-nitride-oxide (O-N-O) sandwich. After resist spin-on, g-line lithography and plasma etching are sequentially invoked to dig a trench approximately 1 um wide and 5 um deep. An arsenic implant of 1.0e+12 cm^{-2} and 200 keV is performed on the trench followed by a diffusion step of 7 minutes and 1000 C. Trench oxidation during the diffusion step is simulated by a 150 A conformal oxide deposition. The trench capacitor is formed by filling the trench with with 0.7 um of N+ polysilicon doped at 1.0e+20 cm^{-3}.

Figures 3.2bc show the SIMPL cross sections after trench lithography and etching. The SAMPLE etch strings used as input to the **Stitch_Back** utility are plotted in Figure 3.2d. Figures 3.2e and 3.2g plot the SIMPL cross sections before and after activation of the arsenic trench implant. The SUPREM-IV mesh used to simulate the implant and diffusion is shown in Figure 3.2f. The complete trench capacitor is shown in Figure 3.2h.

16 Mb DRAM TRENCH PROCESS

- • EPITAXIAL GROWTH
- • DIELECTRIC DEPOSITION
- ○ TRENCH LITHOGRAPHY
- ○ DEEP TRENCH ETCH
- ○ TRENCH IMPLANT & DRIVE-IN
- • TRENCH OXIDATION
- • DOPED POLY FILL

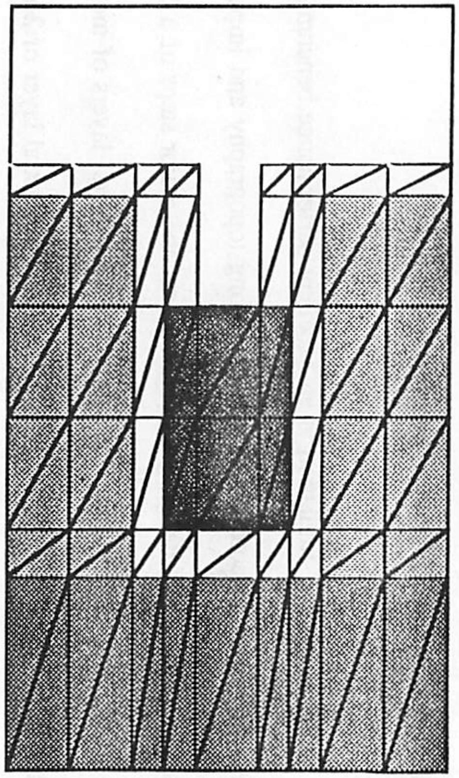
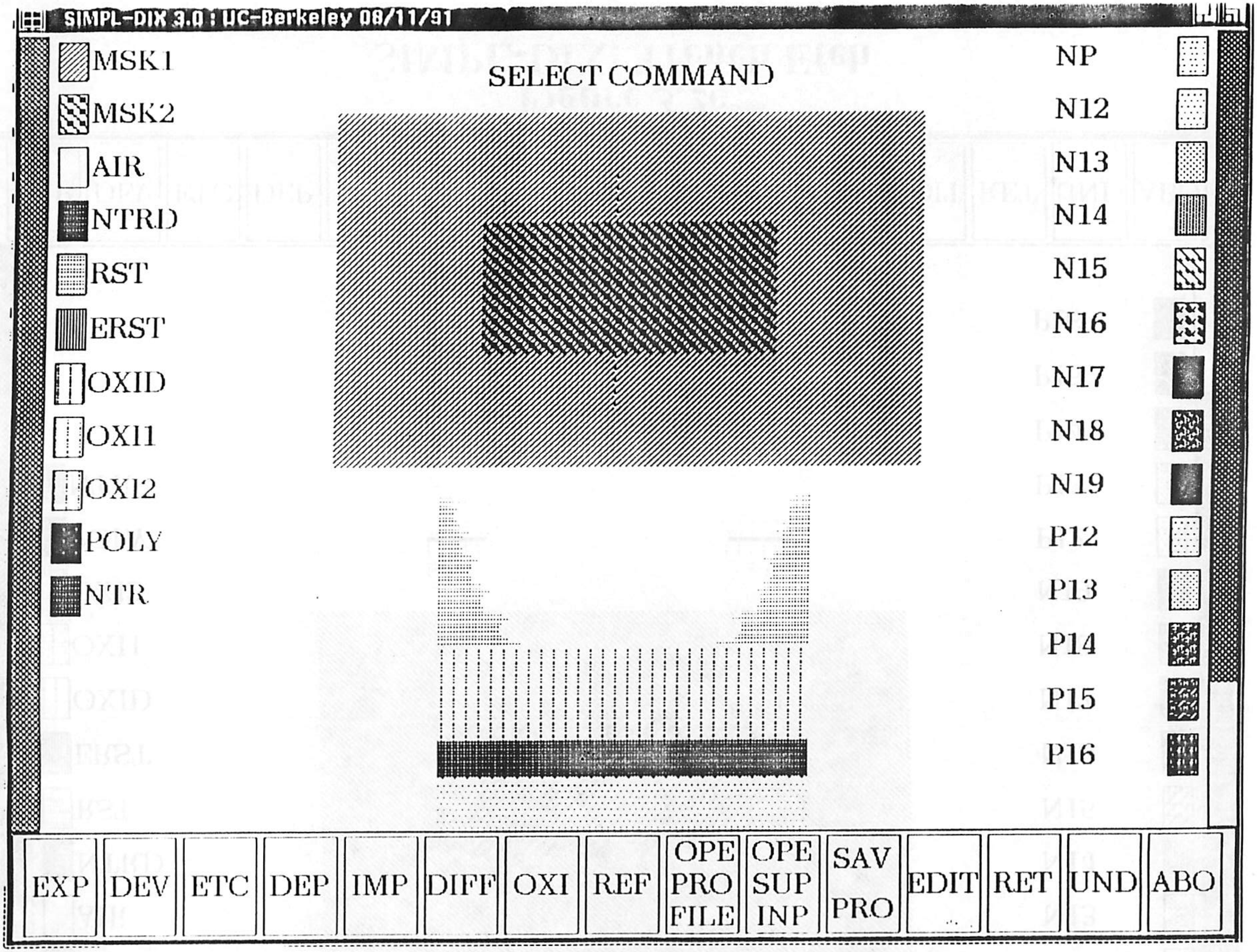
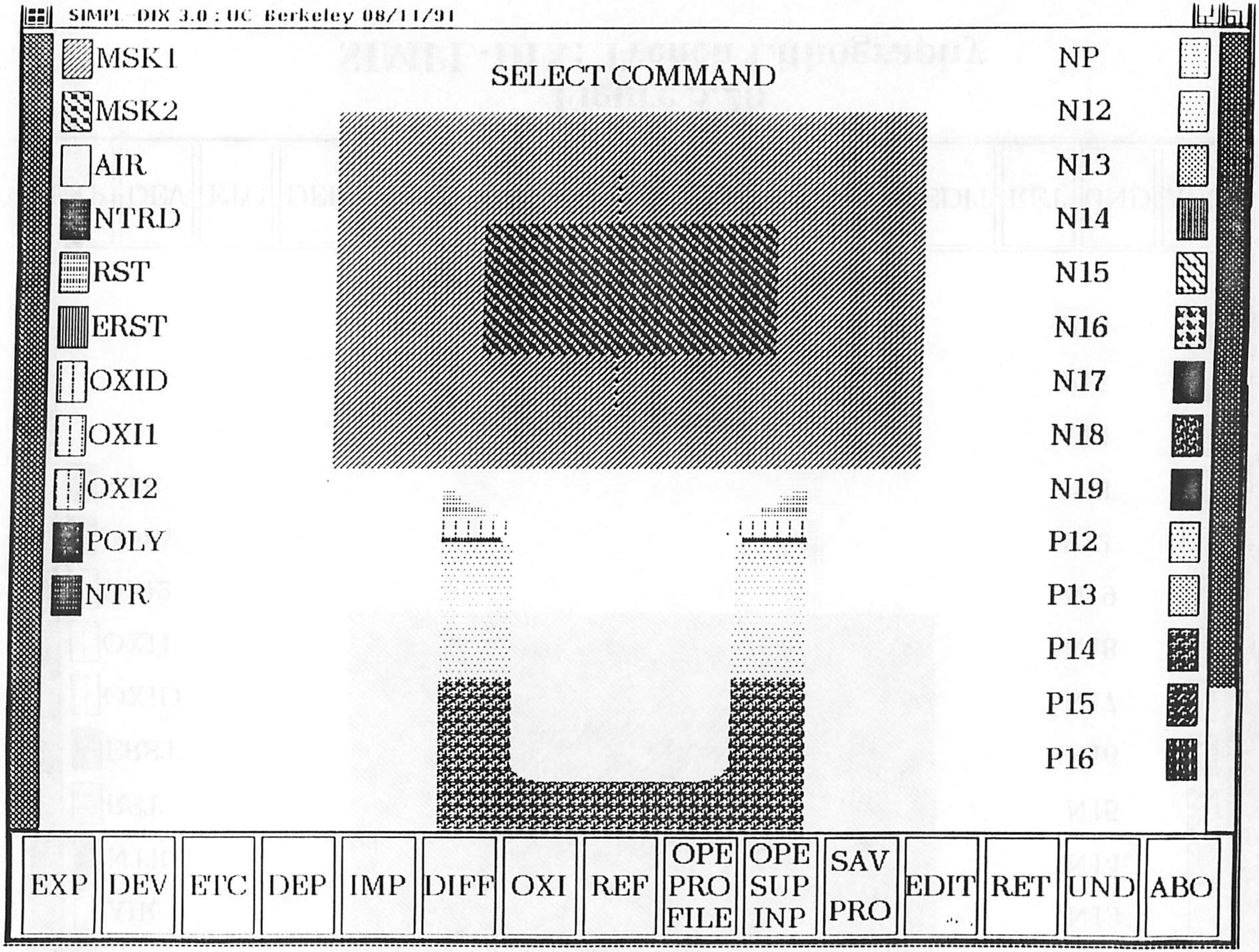


Figure 3.2a


SIMPL-DIX 3.0 : UC-Berkeley 08/11/91

SELECT COMMAND

Figure 3.2b
SIMPL-DIX: Trench Lithography


SIMPL-DIX 3.0 : UC Berkeley 08/11/91

SELECT COMMAND

Figure 3.2c
SIMPL-DIX: Trench Etch

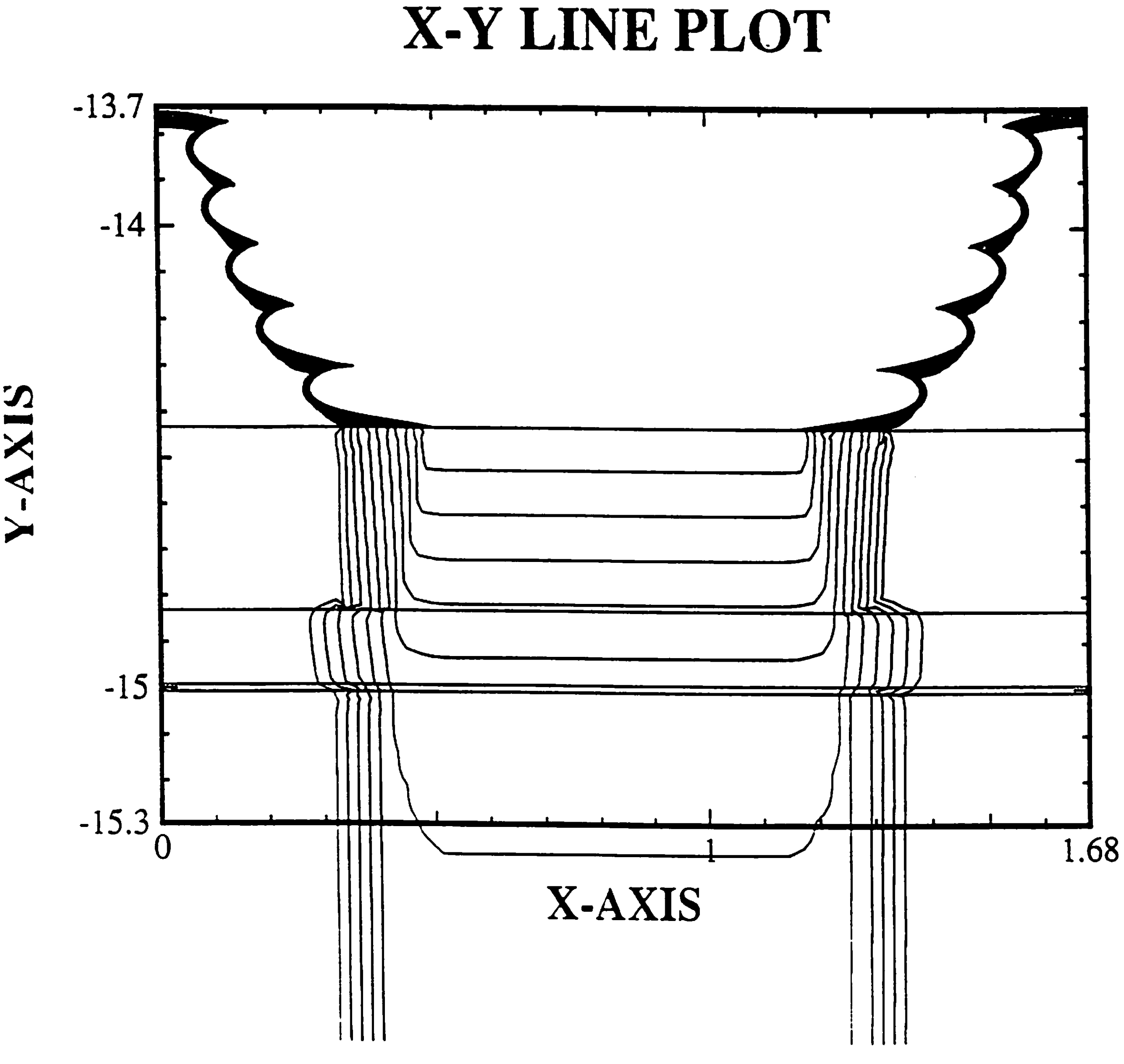
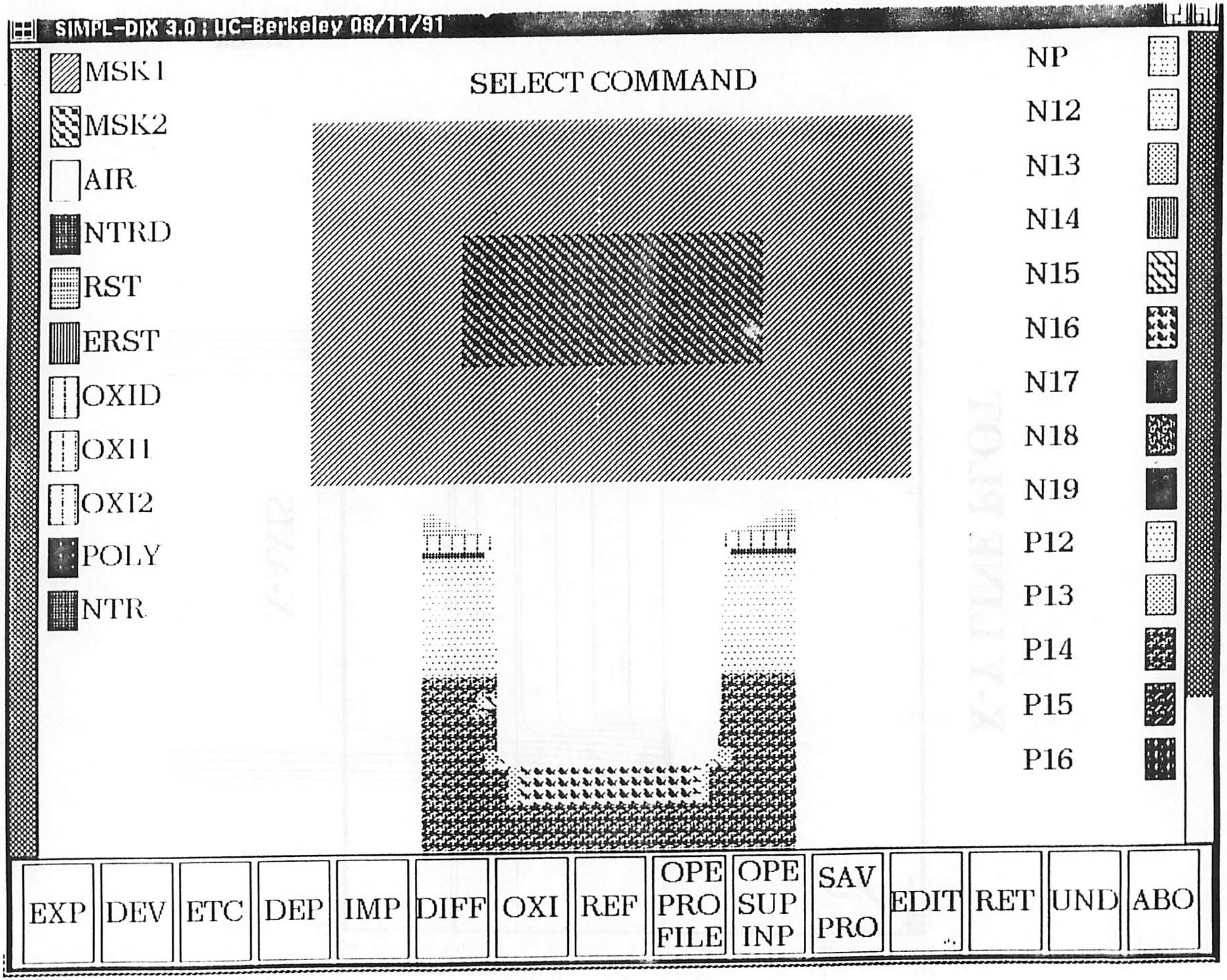


Figure 3.2d
SAMPLE Strings: Trench Etch



SELECT COMMAND

Figure 3.2e
SIMPL-DIX: Trench Implant

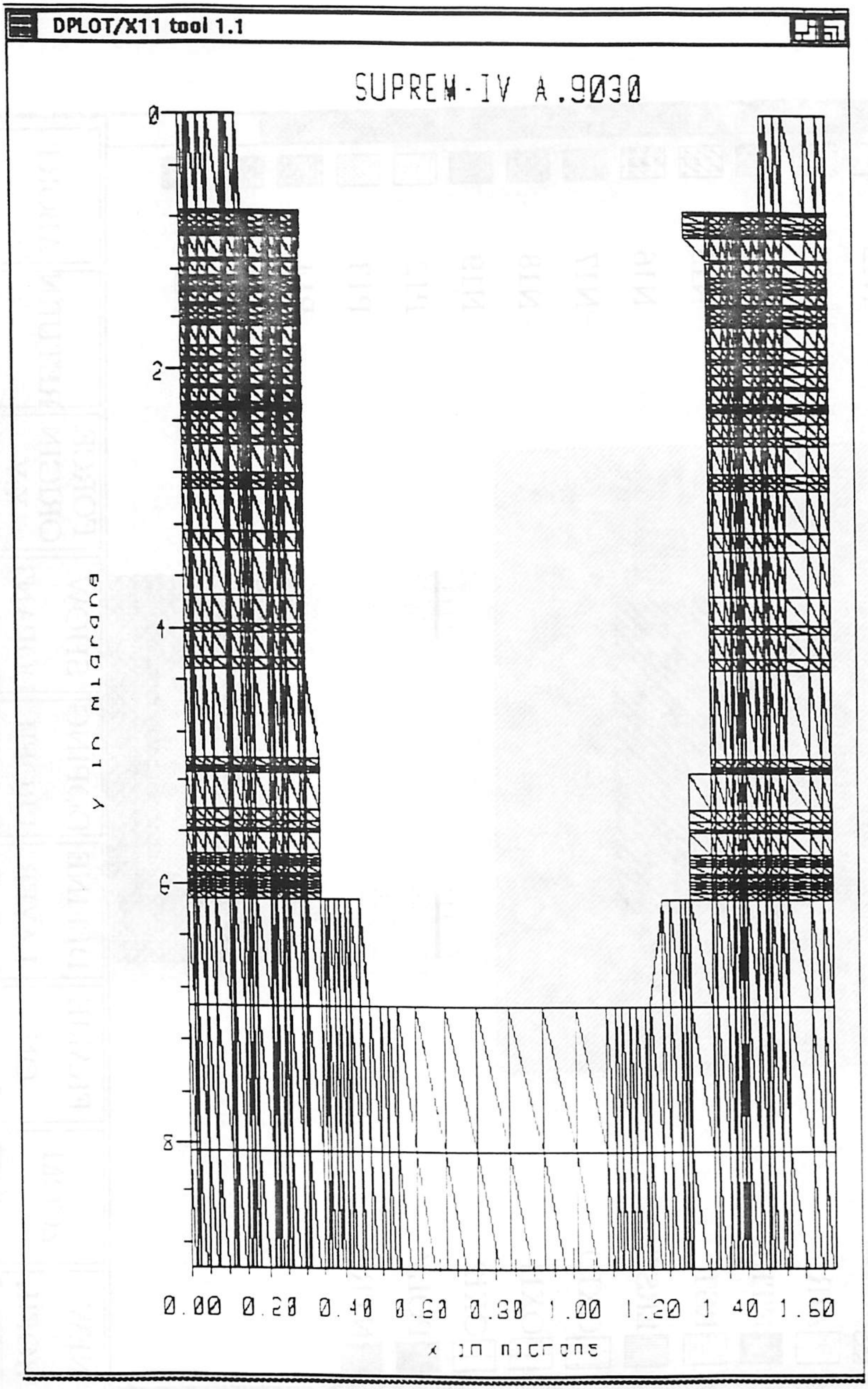
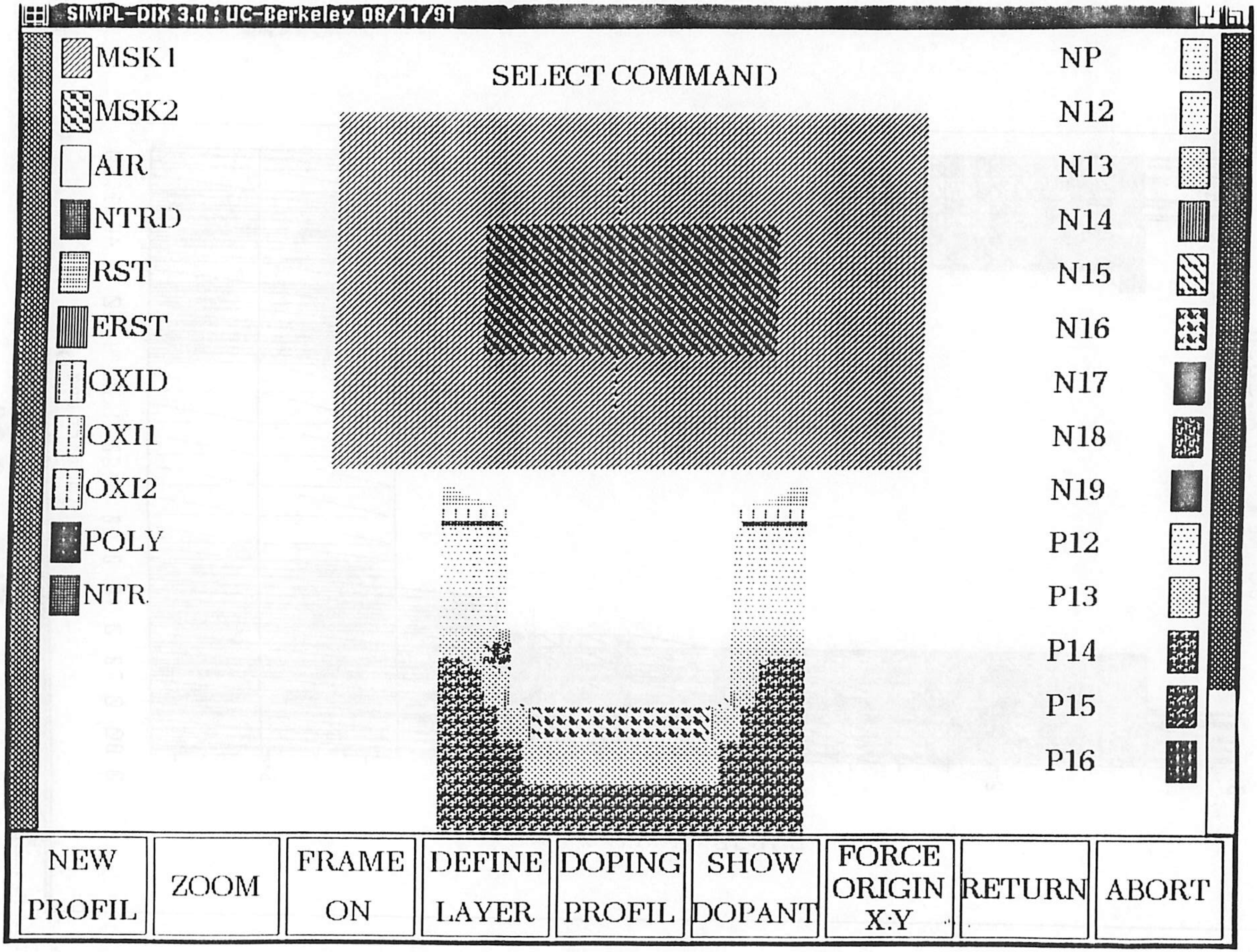


Figure 3.2f
SUPREM-IV Mesh: Trench Implant & Diffusion



SELECT COMMAND

Figure 3.2g
SIMPL-DIX: Trench Diffusion

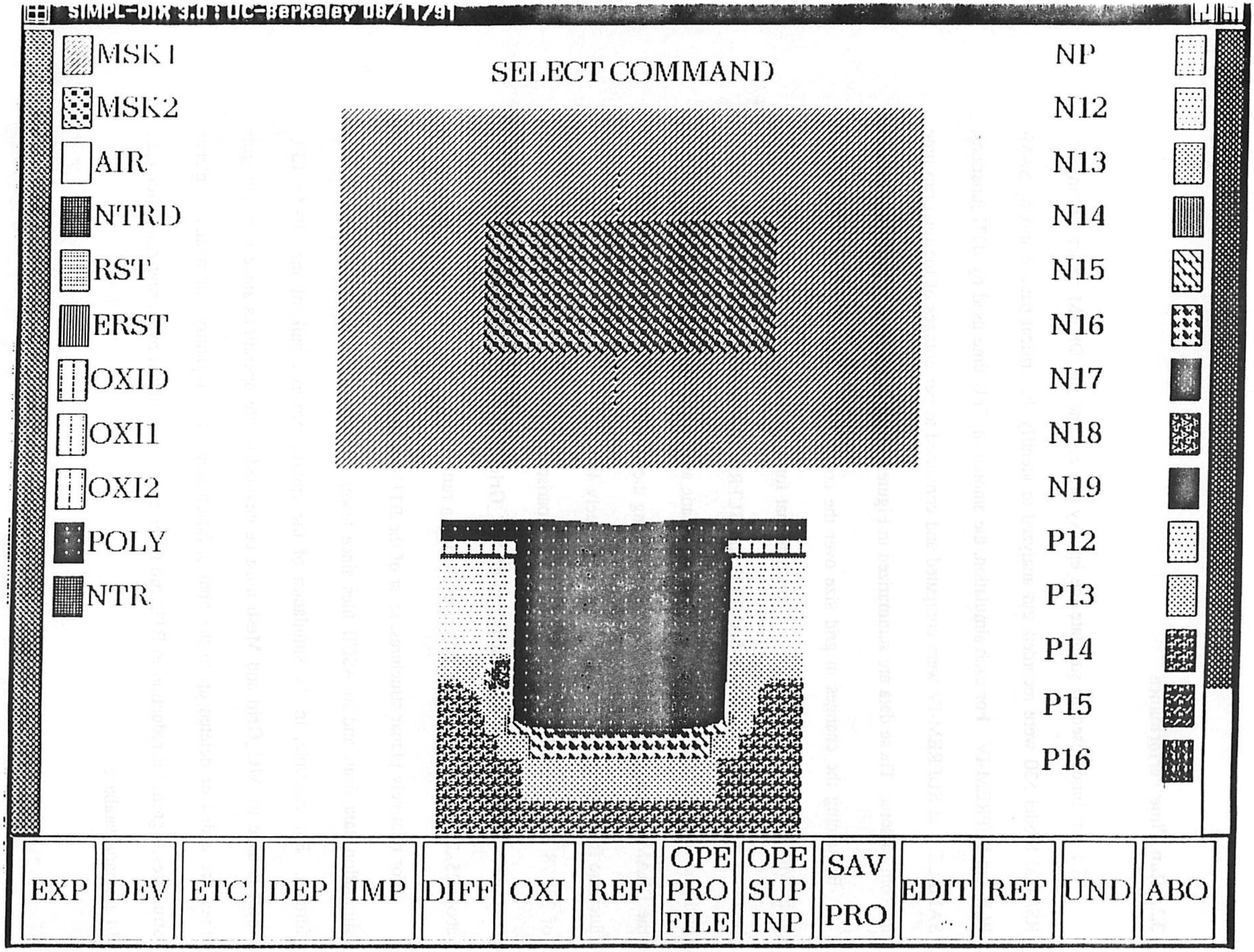


Figure 3.2h
SIMPL-DIX: Trench Capacitor

3.2. Run Time Performance

The run times used to simulate the epitaxy process and the DRAM trench on an IBM RS/6000 Model 530 were recorded and analyzed to identify the critical path in linking SAMPLE and SUPREM-IV. For each simulation, the amount of CPU time used by BTU functions, SAMPLE, and SUPREM-IV were computed and converted to percentages of the total run time for comparisons. These data are summarized in Figures 3.3ab.

By tracing the changes in grid size over the course of each simulation and correlating them with the run time data, one can conclude that for highly nonplanar structures, **MC_Grid** becomes a bottleneck in linking SAMPLE and SUPREM-IV because of the large number of grid line additions and deletions it performs on static arrays. For instance, in the simulation of the DRAM trench, the nonplanar topography along the trench sidewalls caused the grid size to fluctuate from approximately 70x70 to approximately 400x400 twice, and resulted in a run time of 371.78 CPU seconds for **MC_Grid**. By comparison, in the simulation of the epitaxy process, the relatively planar topography caused **MC_Grid** to change the grid size gradually from about 25x25 to about 150x150, which resulted in a run time of only 44.82 CPU seconds.

For relatively planar structures, most of the BTU run time is spent on reading and writing simulation data from and to ASCII files since fewer grid line additions or deletions are performed. For example, in the simulation of the epitaxy process, bulk of the 103.55 CPU seconds used by **MC_Grid** and **Mesh** must be devoted to file operations since very few grid lines are added or deleted up to the final diffusion step. Consequently, for relatively planar structures, significant reduction in BTU run times could be achieved by storing simulation data in a binary database.

Run Time Performance

EPITAXY with BURIED LAYERS

- TOTAL RUN TIME = 1698.29 sec
- BTU Run Time = 103.55 sec (6.10%)
  - MC_Grid = 44.82 sec
  - Mesh = 58.73 sec
- SAMPLE = 1.80 sec (0.10%)
  - Deposition = 1.80 sec
- SUPREM-IV = 1592.94 sec (93.80%)
  - Implant = 86.19 sec
  - Diffusion = 1506.75 sec

Figure 3.3a

Run Time Performance

16-Mb DRAM Trench Process

- TOTAL RUN TIME = 1225.56 sec
- BTU Run Time = 518.43 sec (42.30%)
  - Stitch_Back = 12.40 sec
  - MC_Grid = 371.78 sec
  - Mesh = 134.25 sec
- SAMPLE = 153.86 sec (12.55%)
  - Lithography = 4.48 sec
  - Etching = 149.38 sec
- SUPREM-IV = 553.27 sec (45.14%)
  - Implant = 22.60 sec
  - Diffusion = 553.27 sec

Figure 3.3b

4. Conclusions

4.1. Summary

The primary focus of this project has been the development of a set of functions, BTU, to address the topography and mesh point consistency problem which arises in the linking of topography and impurity diffusion simulations. In addition, a high level procedural interface for accessing BTU functions has been defined to facilitate the extension and maintenance of simulator interfaces and give TCAD developers the option to improve the robustness and efficiency of BTU functions by incorporating geometric modellers and adaptive grid generators. Simulations of epitaxy with buried layers for twin-well CMOS and BiCMOS processes and a 16-Mb DRAM trench capacitor were used to verify the links to SAMPLE and SUPREM-IV. For most structures, BTU string utilities such as **Stitch_Back** require on the order of 10 CPU seconds on an IBM RS/6000 Model 530. The run time of BTU grid and mesh utilities such as **MC_Grid** varies greatly with grid size, which depends directly on topographical complexity. In particular, the **MC_Grid** utility has been identified as a bottleneck in linking SAMPLE and SUPREM-IV for highly nonplanar structures such as the trench,

4.2. Recommendations for Future Work

First of all, the run time of the **MC_Grid** utility needs to be reduced. This can be accomplished by modifying the **MC_Grid** algorithm and changing the rectangular grid data structure. The **MC_Grid** algorithm should be modified to avoid adding grid lines which are likely to be removed in the latter stages of the **MC_Grid** operation. The rectangular grid data structure which **MC_Grid** is implemented on should be dynamic to reduce the computational cost of grid line additions and removal.

Secondly, the **Unmesh** utility should be extended to handle oxidation during impurity diffusion. The extension involves modifying **Unmesh** to incorporate oxidation induced topography changes into the polygon and rectangular grid data structures. This will enable using BTU to facilitate rigorous simulations of process sequences such as LOCOS and trench oxidation.

Finally, the **Clip_Polygon** function used in the **Stitch_Back** algorithm should be improved to handle more robustly etch strings which are close to material interfaces. This will improve the overall robustness of **Stitch_Back** for updating etchback.

References

[1] Conti, P., Hitschfeld, N., and Fichtner, W., "Ω - An Octree-Based Mixed Element Grid Allocator for Adaptive 3D Device Simulation", NUPAD III Technical Digest, pp. 25-26, June 1990.

[2] Oldham, W.G., Nandgaonkar, S.N., Neureuther, A.R., and O'Toole, M., "A General Simulator for VLSI Lithography and Etching Processes: Part I - Application to Projection Lithography", *IEEE Trans. Electron Devices*, vol. ED-26, pp. 717-722, August 1979.

[3] Oldham, W.G., Neureuther, A.R., Sung, C., Reynolds, J.L., and Nandgaonkar, S.N., "A General Simulator for VLSI Lithography and Etching Processes: Part II - Application to Deposition and Etching", *IEEE Trans. Electron Devices*, vol. ED-27, pp. 1455-1459, August 1980.

[4] Law, M.E., *Two Dimensional Numerical Simulation of Impurity Diffusion*, Ph.D. thesis, Stanford University, January 1988.

[5] Rafferty, C.S., *Stress Effects in Silicon Oxidation - Simulation and Experiments*, Ph.D. thesis, Stanford University, December 1989.

[6] Lee, K., *SIMPL-2 (SIMulated Profiles from Layout - Version 2)*, Ph.D. thesis, UC Berkeley, July 1985.

[7] Scheckler, E.W., Wong, A.S., Wang, R.H., Chin, G.R., Camagna, J.R., Toh, K.K.H., Tadros, K.H., Ferguson, R.A., Neureuther, A.R., Dutton, R.W., "A Utility-Based Integrated Process Simulation System", *1990 Symposium on VLSI Technology: Digest of Technical Papers*, pp. 97-98, June 1990.

[8] Scheckler, E.W., *Extraction of Topography Dependent Electrical Characteristics from Process Simulation using SIMPL, with Applications to Planarization and Dense Interconnect Technologies*, MS report, UC Berkeley, December 1988.

[9] Sedgewick, R., *Algorithms, 2nd Edition*, Addison-Wesley Publishing Co., Reading, MA, 1988.

[10] Haken, R.A., Havemann, R.H., Eklund, R.H., and Hutter, L.N., "BiCMOS Process Technology", *BiCMOS Technology and Applications*, Alvarez, A.R., Editor, Kluwer Academic Publishers, Boston, MA, 1989.

[11] Bakeman, P., Bergendahl, A., Hakey, M., Horak, D., Luce, S., Pierson, B., "A High Performance 16-Mb DRAM Technology", *1990 Symposium on VLSI Technology: Digest of Technical Papers*, pp. 11-12, June 1990.