CLRC

Technical Report
RAL-TR-1998-083

The EVEREST Pre-Processor: Version 4.0

J V Ashby R F Fowler and C Greenough

[illegible]

6th January 1999

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

ISSN 1358-6254

The EVEREST Pre-Processor: Version 4.0

JV Ashby, RF Fowler and C Greenough

December 1998

Abstract

In this report we describe the EVEREST Pre-Processor Module which forms part of the EVEREST suite of programs.

The Pre-Processor Module of the EVEREST Suite provides a straightforward way of defining the geometrical structure of simple devices. Three basic mechanisms are provide: primitive commands, BLOCKS commands and STANDARD models. The Pre-Processor also provides commands to define contact surfaces and implant windows to be used by the Doping Profile Module. The module also includes a basic mesh generator and appropriate commands to control the mesh density and distribution.

The EVEREST suite is one of the products of the ESPRIT project EVEREST (ESPRIT 962E-17, *Three-Dimensional Algorithms for a Robust and Efficient Semiconductor Simulator with Parameter Extraction*). EVEREST was a four-year project supported by the European Community under the European Strategic Program for Research in Information Technology (ESPRIT) which is investigating suitable algorithms for the analysis of semiconductor devices in three dimensions, and developing software implementing the most effective of those algorithms.

The original authors of the Pre-Processor Module were N.Ferguson and C.Trinity College, Dublin and J.V.Ashby and C.Greenough of the Rutherford Appleton Laboratory.

A copy of this report can be found at the Department's web site (*http://www.dci.clrc.ac.uk/*) under page *Group.asp?DCICSEMSW* or anonymous ftp server *www.inf.rl.ac.uk* under the directory *pub/mathsoft/publications*

Mathematical Software Group
Department for Computation and Information
Rutherford Appleton Laboratory
Chilton, DIDCOT
Oxfordshire OX11 0QX

Contents

1 Introduction

This manual describes the fourth version of the EVEREST Pre-Processor. It has two main functions: to describe the geometry of a device and to generate a finite element mesh based on this geometric definition.

The pre-processor is a stand-alone package. First, the user provides a geometric description of the device. This may be done in either of three ways. If the device has a standard shape, which is stored in the module's database, then you need only supply the key dimensions of the structure: in all other instances you define the full geometric structure either using the *primitive* commands or the *blocks* commands. The program validates your description and, providing no errors are detected in this description, the mesh generator generates a finite element mesh for the problem. The results are stored in geometry and mesh neutral files ready for use by the other EVEREST Modules.

2 User Interface

The modules use an interactive command decoder, developed at the Rutherford Appleton Laboratory, with an on-line help facility. This command environment allows you to input the device description and the mesh refinement either interactively or by means of prepared data files which contain this information. The second mode of use is strongly recommended, especially if you are describing a device which will be used repeatedly.

The initial commands available to the user are given by the `HELP` command as shown below. Full details of all the commands available in *prepro* are given in the Appendixes.

```
Model: Help

The commands currently defined are:

Applications Commands

 GEOMETRY -  to enter the geometric modeller
 MESH     -  to enter the mesh generator
 STANDARD -  to define a standard device
 BLOCKS   -  to enter the block geometric modeller
 END      -  to exit the modeller

Internal Commands

 MORE     -  to disply the contents a file
 CHANGE   -  to change working directory
 RENAME   -  to rename a file
 COPY     -  to copy a file
 ERASE    -  to delete (remove) a file
 LIST     -  to provide directory listing
 WRITE    -  to provide monitoring of a session
 READ     -  to redirect the input stream to read from a file
 SYNTAX   -  to provide the syntax of a command
 HELP     -  to access HELP system

For further information type: HELP <command name> [<option>],
where <option> is BRIEF or FULL
```

All the commands can be typed in upper or lower case. The syntax of each command can be obtained by using the `syntax` command. For example

```
Geometry: syntax write

WRIte STAte=<choice> [,FIle=<string>] [,PROmpt=<choice>]

Geometry:
```

To get full details on a command and its parameters, such as the `WRITE` command, you can use `HELP WRITE`.

```
Model: help write

 Name     : WRITE
 Purpose  : to provide monitoring of a session
 Syntax   : WRIte STAte=<choice> [,FIle=<string>] [,PROmpt=<choice>]

 Keyword    Type             Status        Current Value
 -----------------------------------------------------------------

 STATE      choice           required       on,off,close
 FILE       string           retained       MONITOR
 PROMPT     choice           reset          on,OFF
```

A command can be abbreviated, the shortest value being indicated by the uppercase letters in the syntax section, e.g. the `Write` command may be shortened to just `wri`. The system is reasonably simple to use and working through one or two of the examples below should enable one to get to grips with it.

Prepro provides a number of *internal* commands. These commands, such as `HELP` and `COPY`, provide standard information and file handling from within *prepro*. As with all commands, details of their usage can be obtained through the `HELP` command. A summary of these commands is given in the Appendix A.

Those commands that access the file store do so by invoking the appropriate system command of the operating system being used. This means that in general if a report or an error on an action is produced, these will be those of the host operating system.

For example, on UNIX systems, the `RENAME` command will use the UNIX command `mv`. Similarly `LIST` uses the UNIX command `ls`. Although the parameter types for these commands is *string*, the appropriate host systems file expression and options can be used provide any expression that contain spaces any contained within quotation marks. An example of this is:

```
    LIST "-l *.MSH"
```

On a UNIX system this command will list all files with extension `.MSH` in the current working directory.

3 Description of Device Geometry

The commands to describe the device geometry and those to specify the mesh refinement are entered in separate levels within *prepro*. The commands GEOMETRY, STANDARD and BLOCKS enter the three levels which provide geometry definition commands.

3.1 Device description using the GEOMETRY commands

The geometry input module reads the geometric description of the device supplied by the user. The device is described in terms of basic primitives such as points, lines, surfaces and volumes. In addition, it is possible to describe the position of windows used in the injection of impurities. Contacts are described as collections of surfaces and windows. At this stage of the program the commands are checked for syntactic correctness.

The geometry input commands should be used in the description of simple, regular structures only. Although in principle there is no reason why the commands may not be used in the description of more complicated, irregular structures this can be very error prone. The commands and examples of their use are given in Appendix B.

The commands for the geometry level are:

POINT	- to define a point used in the device description
LINE	- to define a line used in the device description
SURFACE	- to define a surface used in the device description
VOLUME	- to define a volume used in the device description
WINDOW	- to define a window used in the device description
CONTACT	- to define a contact used in the device description
NEUTRAL	- to specify the output geometry neutral file
RECAP	- to review the commands entered so far
CLEAR	- to clear all prior geometry commands
GEOM	- to verify the geometry data
QUIT	- to exit the geometry section without processing the user-supplied data
END	- to end the input and process the data.

A device is described in terms of geometric primitives: points, lines, surfaces and volumes. A window is a sub-surface which may either be part of a contact or may define a surface area through which impurities are introduced to the silicon. You can specify lines of symmetry for the profile description in the definition of the window. A contact is defined as being a non-empty collection of surfaces and windows.

Below is given some of the commands required to define a simple one-dimensional diode constructured from two blocks of silicon as shown in Figure 1. The descriptions are not complete - below and in Figure 1 but serve to illustrate the method.

```
geometry
 point p1 5 0 0
 point p2 0 0 0
 point p3 0 0 2

 . . . . . . .
 . . . . . . .
```

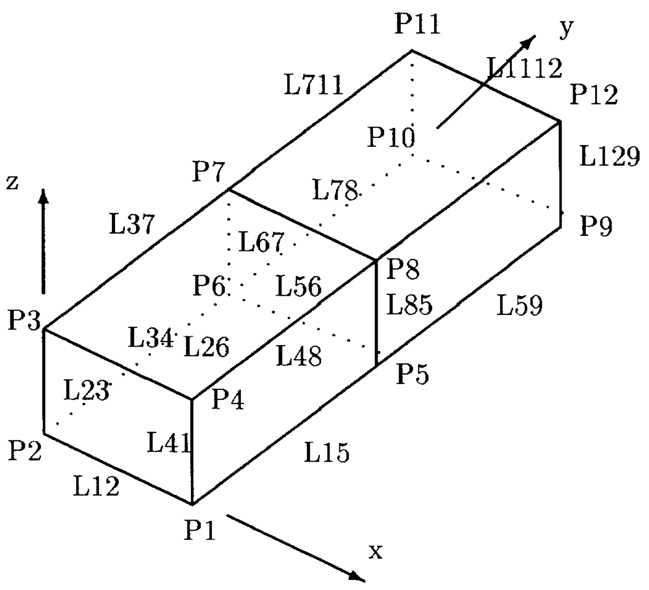

Figure 1: Description on a *p-n* diode

```
po p9 5 20 0
po p10 0 20 0
po p11 0 20 2
po p12 5 20 2

line l12 (p1,p2)
line l23 (p2,p3)
line l34 (p3,p4)
line l41 (p4,p1)

. . . . . . .
. . . . . . .

li l37 (p7,p3)
li l59 (p5,p9)
li l912 (p9,p12)
li l1112 (p12,p11)

surface s1 (l12,l23,l34,l41)
suface s2 (l56,l67,l78,l85)
suface s3 (l910,l1011,l1112,l129)

. . . . . . .
. . . . . . .

su s9 (l78,l711,l1112,l812)
su s10 (l85,l812,l129,l59)
su s11 (156,1610,1109,159)

volume v1 (s1,s2,s4,s5,s6,s7) silicon
vol v2 (s2,s3,s8,s9,s10,s11) silicon

contact c1 s1
con c2 s3

neutral p-n

geom
```

Other commands allow you to specify the neutral file to which the information is written and to recap the commands entered so far. Finally, you may exit the geometric modeller in either of two ways: first, the device description is validated before proceeding to the mesh generator (GEOM); secondly, you may choose to exit the modeller and proceed no further with the session (QUIT).

3.2 Device definition using STANDARD Models

You are supplied with a library of standard models. If the device you wish to model has the same shape as one of the models, all you need do to describe the device is specify the key dimensions of the device, e.g. the device length, breadth and height. These commands may be accessed by selecting the STANDARD command level when using the pre-processor.

At the moment there are three models in the library. The first is a MOSFET device; the user completes its definition by specifying ten distances. The second is a one-dimensional diode, which is specified by defining four distances. The third is a polysilicon bipolar emitter, which is specified by defining nine distances. It is possible to augment the contents of the library yourself. In Ferguson and Fitzsimons (1988), the authors explain how to add another model to the library; they also explain how the EVEREST code must be modified to include the new model. While it is possible to add to the library in this manner, the preferred method for adding a model to the library is to specify the model completely and send the specification to those maintaining the code.

The principle on which the standard models library is based is quite simple. The topological description of the device in terms of its lines, surfaces, volumes, contacts and windows is pre-checked and stored in a database. When the user supplies the dimensions, the point co-ordinates are determined and checked for consistency. Then all other information is retrieved from the database and the geometric model is complete. This results in a reduction in time spent in the geometry input phase.

The Standard Models section of the pre-processor provides ready access to a database of standard device shapes. Instead of providing a full description of the device, you supply the main dimensions of the structure as defined in the command reference section in Appendix C. The other commands allow you to specify the neutral file to which the geometric description is written and to exit the standard modeller, either to finish the session or to proceed to use the mesh generator. The commands for the standard model generator are:

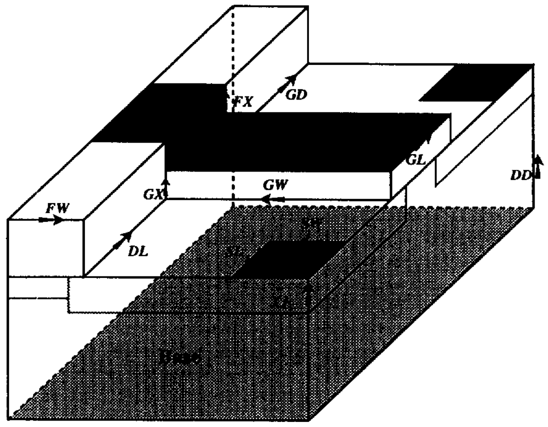

Figure 2: The Standard MOSFET

`NEUTRAL`	- to specify the neutral file to which the output is to be written.
`STANDARD`	- to end the input and process the data.
`QUIT`	- to exit the Standard Models section without processing the user-supplied data.
`MOSFET`	- to define a standard MOSFET by giving its dimensions.
`ONE_DIODE`	- to define a standard one-dimensional diode.
`BIPOLAR`	- to define a standard bipolar polysilicon emitter.

Figure 2 show the structure of the standard MOSFET together with the basic control parameters. All the sizes labled can be specified through the `MOSFET` commands as shown in the two examples below.

```
MOSFET GL = 2, GW =  1.5, DD = 2, GX = 0.2, FX = 0.05, XJ = 0.3,
       SL = 0.5, SW = 0.2, DL = 1, FW = 0.5
MOS 3 1 4 0.05 0.0 0.8 1 0.3 0.25 0.25
```

3.3 Device description using BLOCKS commands

This section describes the user interface for the BLOCKS module of the EVEREST geometric modeller. The prime consideration in developing the interface has been to generate a set of commands, which is capable of defining geometries of most semiconductor devices with a minimum of input, yet with the flexibility to define more complex geometries.

The commands which define a device geometry are limited to VOLUME, CONTACT and WINDOW. Each of these uses simple right angled primitives, HEXAHEDRON, PRISM and TETRAHEDRON in the VOLUME command and RECTANGLE and TRIANGLE in the CONTACT and WINDOW commands, all of which are defined by cartesian coordinates of two or three points. All complex geometries, even non right angled, can be defined by combinations of these primitives. The commands and examples of their use are given in Appendix D.

3.3.1 Volume definition

The VOLUME command is used to define volumes in space by specifying their shape, position, extent, material and, where necessary, orientation. The BLOCKS module automatically generates all the relevant points, lines and surfaces to translate this specification into the hierarchical description used by the GEOMETRY module, the neutral files and the subsequent modules of the EVEREST suite.

Three volume primitives are recognised: brick, prism and tetrahedron. In suitable combinations these can be used to build most device geometries. They are limited to right-angled forms of these shapes with the (solid) right-angles oriented parallel to the coordinate axes. This may mean that some ingenuity is required to specify particularly complex geometries but for most structures used in device modelling this will be adequate.

Each of the volumes is given a name and a material. These must be specified as parameters to the VOLUME command. All the subsidiary entities (points., lines and surfaces) will have names generated automatically using the convention Pxxx, Lxxx and Sxxx where xxx is a three digit number with leading zeroes where necessary.

The brick primitive is a simple cuboid oriented parallel to the axes. It is described by the position of *any* of its vertices and the (signed) distances to each of the three neighbouring vertices. An alternative way of looking at these distances is as the body diagonal vector from that vertex (see Figure 3).

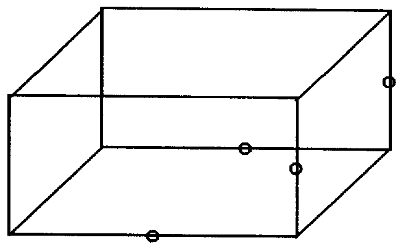
Figure 3: The BRICK volume primitive

The prism primitive is a right-triangular prism with its triangle lying in one of the principal planes (x-y, x-z or y-z). It is described by the position of *either* of its right-angled vertices and the (signed) distances to each of the three neighbouring vertices. This alone is not enough to specify a prism completely. To complete the definition the orientation must also be given. This is the name of the axis parallel to which the major axis of the prism lies. Thus a prism with x-orientation has its triangles lying in the y-z plane (see Figure 4).

The tetrahedron primitive is a tetrahedron, three of whose triangles are right-angled with the three right-angles meeting at the same vertex. In addition each of the right-angled triangles lies in a principal plane. It is described by the position of the *single* right-angled vertex and the (signed) distances to each of the three neighbouring vertices (see Figure 5).

3.3.2 Window definition

The pre-processor recognises two types of windows; those used for dopant definition and those used in the specification of contacts. Since the CONTACT command does away with the need for the user

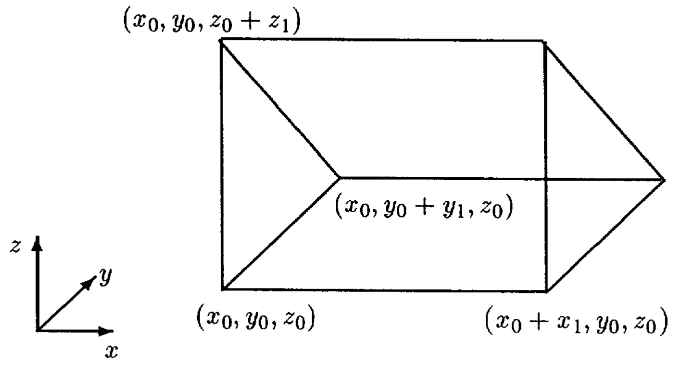

Figure 4: The PRISM volume primitive

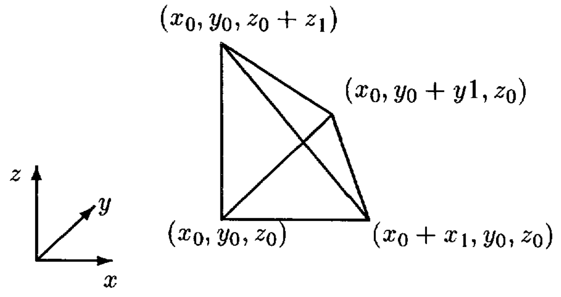

Figure 5: The TETRAHEDRON volume primitive

to know about which surfaces and parts of surfaces a contact covers the `WINDOW` command is used *only* to define the former type.

A window is a plane region coincident with some surfaces of the device. It can be either a rectangle or a triangle. In either case it is specified by three points whose coordinates are given in a real list of nine entries. See Figure 6 for details of this.

The doping profile generator requires information on how to treat the boundaries of windows, either as lines of symmetry or not. This information is passed through the `LSYMM` parameter to the `WINDOW` command. Figure 6 illustrates the convention used by this parameter.

3.3.3 Contact definition

A contact is a plane region coincident with some surfaces of the device. It can be either a rectangle or a triangle. In either case it is specified by three points whose coordinates are given in a real list of nine entries. See Figure 6 for details of this.

It is not necessary for a contact to lie within a single surface. If it overlaps more than one `BLOCKS` will automatically generate points and lines for the intersections and windows for any partially covered

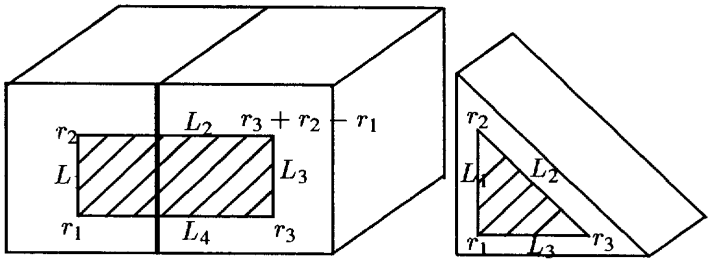

Figure 6: Two windows, one rectangular and one triangular. Note a) the positions of r_1, r_2 and r_3 on the rectangular window and b) the order of the lines L1–L4 for the use of LSYMM

surfaces. These automatically generated windows cannot be used for doping purposes except where they coincide with a window defined with the WINDOW command.

3.4 Generation of the Geometry

The GENERATE command is used to combine all the geometric data for the primitives into a single, consistent description of the whole device. The first operation is to eliminate redundant points; for instance, defining two unit cubes, one at $(0,0,0)$ and the other at $(1,0,0)$ will result in the initial definition of twelve points, four pairs of which coincide (at $x = 1$). The later ones will be eliminated.

Then the lines will be similarly checked. Again, in the case of the two cubes four pairs of lines coincide. Then the surfaces are checked and the single pair of coincident surfaces will be reduced to one surface. Finally in this phase the windows are checked and redundancies eliminated.

The next phase checks the contacts for redundancy and then proceeds to generate any new windows which may be required (together with their associated points and lines). It also matches contacts to surfaces. Following this the first phase is repeated to remove any new redundancies which might have been introduced.

4 Mesh Generation

The commands necessary to generate and refine a mesh for a valid geometry are listed below. Some of these are useful if you are entering the commands interactively. The RECAP command allows you to check on the commands entered so far. If you have made a mistake then the CLEAR and QUIT commands allow you either to restart entering the data or to leave the mesh generator.

The first steps are to use the GLOBREF and REFINE commands to specify the point density within the geometry and on specific lines. The mesh of the device is then generated with the END and the MESH commands. The MESH command returns to the input section and preserves the previously defined refinements, allowing you to modify them if necessary. The commands for the mesh generator are:

CLEAR	- to clear memory and allow the user to re-enter mesh data.
END	- to end the mesh input, generate the mesh and exit.
GLOBREF	- to specify the default mesh refinement.
MESH	- to end the mesh input, generate the mesh and return to the mesh input section.
MULREF	- to specify refinement on multi-pointed lines.
NEUTRAL	- to specify the name of the mesh neutral file.
QUIT	- to leave the mesh input section immediately.
RECAP	- to review the mesh data entered.
REFINE	- to specify mesh refinement.
SREF	- to specify mesh refinement for standard models.
QUERY_LINE	- to find line names along a given axis.

Before a mesh can be generated the level of mesh refinement should be specificied. This can be done by using the GLOBREF or REFINE commands. GLOBREF defines a default mesh refinement. For example GLOBREF 5 would specify 5 subdivisions on all lines. The REFINE command allows more careful descriptions of refinement.

The REFINE command is used to refine a line, which you have already defined in the geometry or block section. You specify the number of subdivisions on the line and a weighting of the subdivisions, this is a real number in the range -1 to $+1$. If the weighting is negative then the points will be weighted towards the first defining point of the line. If the weighting is positive then the points will be weighted towards the second defining point of the line. If the weighting is zero then the points will be generated uniformly on the line. A similar command, MULREF, is used to refine multi-pointed lines. The number of subdivisions and their weighting are entered in list form, with the i-th entry in each list defining the refinement on the i-th section of the line. For geometries generated by the BLOCKS level the QUERY_LINE command can be used to find the line name to refine.

After the refinement commands have been entered and validated, control passes to the mesh generation routines. This part of the program contains the following components :

- Adding and possibly modifying refinement to all lines in the device.
- Orienting the volumes so that they are consistent.
- Generating nodes on lines and surfaces.
- Generating nodes inside volumes.
- Generating hexahedral elements.

5 Output from prepro

To facilitate the post-processor it is useful to know which elements lie on which surfaces. For each face of the generated hexahedra the system stores a pointer to the geometric surface containing that face, provided the face is on a surface. At this stage of the program this information is reordered to produce for each surface, a list of elements and their faces lying on that surface. This is then written to the neutral file in a \$SURF group. Note that there will be one \$SURF group for each surface.

As part of the post-processing, the monitor statistics are collated and output at the end of the reference file MESH.TIM.

After completion of the geometry and mesh generation routines, the data is output to the neutral files. You may specify the name of the neutral files in the geometry and mesh input sections. The following groups are output to the neutral file in the geometry section: $POIN, $LINE, $FACE, $VOLU, $COGE, $CONT and $WIND. In the mesh section the following groups are output : $NODE, $NTYP, $ELEM, $SURF and $NBVO.

6 Installation Details

The Geometry and Mesh Modules requires certain information about the host computer to function properly. The geometry routines use the same I/O streams as the command decoder. The parameters which govern the complexity of device description which the code can handle are in the insert file `param.common`. They are

NPONT	- the maximum number of points allowed.
NLINE	- the maximum number of lines allowed.
NSRFC	- the maximum number of surfaces allowed.
NVLME	- the maximum number of volumes allowed.
NCNTC	- the maximum number of contacts allowed.
NWNDW	- the maximum number of windows allowed.

The other parameter of which the installer should be aware, is the parameter `EPS`. This is contained in the file `geom.param` and is used as the error tolerance when checking the numerical data, e.g. to determine whether two line segments intersect.

The mesh routines require that the following parameters (presently stored in the insert file, `mesh.param`) be given values consistent with your system and mesh generation requirements. These parameters are:

MAXNOD	- the maximum number of nodes allowed.
MAXELT	- the maximum number of tetrahedral elements allowed. This must have a value in excess of $5(MAXNOD^{1/3} - 1)^3$.
MAXDIV	- the maximum number of subdivisions allowed with the `REFINE`, `GLOBREF` and `SREF` commands.
REFILE	- unit number of the file `MESH.REFER`.
INTERM	- unit number of terminal input.
OUTERM	- unit number of terminal output.
EPS	- smallest positive number such that $1 + EPS \neq 1$.

The two machine dependent functions, `CPUTIM` and `DATTIM`, may need to be changed. The double precision function `CPUTIM` returns the current CPU time. The character function `DATTIM` returns the current date and time in CHARACTER*20 format.

7 Algorithms

Many of the checks for consistency of the user's geometry specification involve the comparison of all pairs of elements in a set of data. This need not be detailed further. The routines for checking the consistency of the point and line data use algorithms of this type only. There is nothing of interest in the routines which read in your input and write out the results. The other routines are discussed in turn.

The routine to check the surface definitions tests that the boundary of the surface is simply connected. This is done by testing each pair of line segments in the definition of the boundary and seeing if they intersect at a point other than an end-point. This involves calculating the parametric equation of each line segment and comparing the parametric equations. The parametric equation used is

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1} = t \tag{5.1.1}$$

for a line through the two points (x_1, y_1, z_1) and (x_2, y_2, z_2). Special care must be exercised in the event of any of the denominators vanishing. Although in general this seldom happens, owing to the restrictions on the geometry in the first phase of the code development, this is the norm rather than the exception. The program also checks that the surface is planar. Given three points, $\mathbf{a}, \mathbf{b}, \mathbf{c}$, in three-space the equation of the plane they define is given by

$$(\mathbf{x} - \mathbf{a}) \,.\, \mathbf{\Phi} = 0 \tag{5.1.2}$$

where

$$\mathbf{\Phi} = \mathbf{b} \times \mathbf{c} + \mathbf{c} \times \mathbf{a} + \mathbf{a} \times \mathbf{b}$$

When validating the volume input checks are made for holes in the device. This is done by computing Euler's formula for the device. The formula is

$$N_0 - N_1 + N_2 - N_3 = 1 \tag{5.1.3}$$

where N_i denotes the number of i-dimensional entities. (See Finney, 1979.) If any windows are used in the description of the device, it is no longer a simple matter of using the total number of points, lines, surfaces and volumes defined, because the window definition uses points and lines to define subsets of surfaces. Thus it is necessary to determine the number of points used in the definition of windows only, and the number of lines used in the definition of windows only. If these two quantities are denoted by M_0 and M_1 respectively, then the modified Euler formula becomes

$$(N_0 - M_0) - (N_1 - M_1) + N_2 - N_3 = 1 \tag{5.1.4}$$

where N_i is understood to mean the total number of i-dimensional quantities defined by the user.

To check that a window lies on a given surface (5.1.2) is used, where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are chosen to be three, non-collinear, points defining the surface in question. Each defining point of the window is plugged into the equation to complete the test. When checking the window definitions, the defining lines are put into cyclical order. They must also be in anti-clockwise order when viewed from outside the device. This last condition is satisfied by finding a point on the side of the volume opposite the window and viewing the window from there; if the lines are in anti-clockwise order then reverse them. (Remember that you're viewing from inside the device.)

Checking that a contact is connected is not as straightforward as it might seem. It is not just a case of testing that each defining surface has a line in common with at least one other surface. The matter is complicated by the fact that contact definitions may comprise both surfaces and windows. A line which defines a window is usually a subset of a line defining a surface. Therefore the test for line intersection must be performed on a new collection of lines. To check that two contacts don't touch, it suffices to determine their defining points and ensure that they have no points in common.

In the 'post-processing' of this data there is nothing remarkable about the way in which the neighbour table of volumes is built up. The order in which material types is assigned to the user-defined entities is important. First, all entities used in the definition of contacts are labelled. Then all unlabelled entities which are on interfaces between different materials are labelled as being on the given interface. Finally, all unlabelled entities are labelled as being of the same material type as the volume they help to define.

8 Some Examples

Three examples are discussed. These show the versatility of the preprocessor in this release.

8.1 Example 1: A Corner Diode

The first example is that of a 'corner' diode, i.e. one in which a small region of p-doped silicon is inset in a block of n-doped silicon (or *vice versa*). This example is chosen because, although the device structure is simple, with the present restrictions on the descriptions of devices it is necessary to use eight volumes to describe the device. Also the example uses the window command to define one of the contacts. The device is shown in Figure 7 and its geometric description is given in Appendix F. The refinement commands for meshing the structure are also given in that appendix.

8.2 Example 2: A Three-dimensional Diode

This example is the device defined as one of the benchmark problems for the code. The device is simpler to describe geometrically than the first example because not all the regions have constant doping. However, in order to facilitate the mesh generator, the block of silicon has been split into three layers in the definition. From the point of view of geometric description, the interest in this device lies in the non-planar contact and the oxide region.

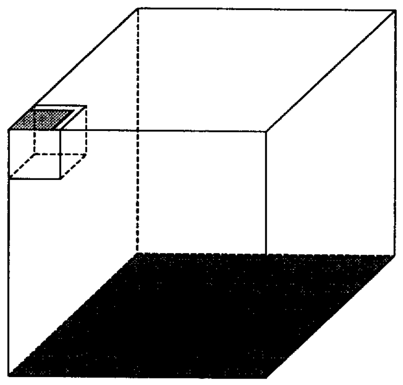

Figure 7: The Corner Diode

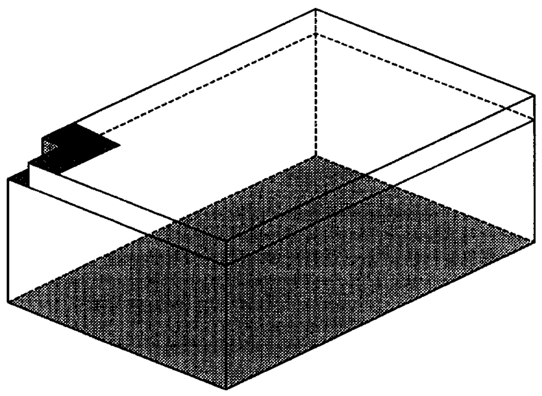

Figure 8: The Three-Dimensional Diode

The non-planar contact is interesting because it requires many surfaces for its definition. The device is shown in Figure 8 and its geometric description is given in Appendix F; this appendix also contains the refinement commands for meshing the structure.

8.3 Example 3: A Standard MOSFET

A standard MOSFET has been chosen as the third example. All you need supply are the dimensions of the device using the MOSFET command in the STANDARD section of the pre-processor. Given these dimensions, all the points are generated automatically by the code and all data is retrieved from the database regardless of the dimensions of the device. The program automatically accounts for a field oxide of zero thickness by discarding the volume, surface, and line commands which describe the field oxide region.

The shape of the device is shown in Appendix C, in Figure 10 and the geometric description of the device is given in Appendix F, along with the commands for meshing the structure.

9 References

Ferguson N., Fitzsimons C.J."On the Use of Standard Models in EVEREST" *ESPRIT Project 962 Report* April (1988)

Finney J.L. "Random packings and the structure of simple liquids. I. The geometry of random close packing" *Proc. Roy. Soc. Lond. A.* **319** 479–493 (1970)

A Internal Commands

A.1	MORE	to display the contents a file
A.2	CHANGE	to change working directory
A.3	RENAME	to rename a file
A.4	COPY	to copy a file
A.5	RM	to delete (remove) a file
A.6	LIST	to provide directory listing
A.7	WRITE	to provide monitoring of a session
A.8	READ	to redirect the input stream to read from a file
A.9	SYNTAX	to provide the syntax of a command
A.10	HELP	to access HELP system

A.1 MORE - to display the contents a file

Syntax

```
MORe FILe=<string>
```

Description

Displays the contents of the specified file in the current window.

Parameters

FILE	Required *string* A *string* giving the name of the file to be displayed.

Examples

```
more file=ANODE
MOR CATHODE
```

A.2 CHANGE - to change working directory

Syntax

```
CHAnge DIRectory=<string>
```

Description

Changes the current working directory.

Parameters

DIRECTORY	Required *string* A *string* giving the name of the new working directory.

Examples

```
change directory=results
CHA MODELS
```

A.3 RENAME - to rename a file

Syntax

```
REName FILe1=<string> FILe2=<string>
```

Description

Renames a given file to a new name.

Parameters

FILE1	Required *string* A *string* giving the current file name.
FILE2	Required *string* A *string* giving the new file name.

Examples

```
RENAME FILE1=RESULT1 FILE2=RESULT.SAVE
ren output1 output2
```

A.4 COPY - to copy a file

Syntax

```
COPy FILe1=<string> FILe2=<string>
```

Description

Copies a given file to a new file.

Parameters

FILE1	Required *string* A *string* giving the source file name.
FILE2	Required *string* A *string* giving the destination file name.

Examples

```
COPY FILE1=RESULT1 FILE2=RESULT.SAVE
cop output1 output2
```

A.5 RM - to delete (remove) a file

Syntax

```
RM FILe=<string>
```

Description

Removes (deletes) the given file from the file system.

Parameters

FILE	Required *string* A *string* giving the name of the file to be removed (deleted).

Examples

```
RM FILE=RESULT
rm output
```

A.6 LIST - to provide directory listing

Syntax

```
LISt [FILe=<string>]
```

Description

Provide a listing of the current or specified directory.

Parameters

FILE	Optional *string* : initial = " " A *string* giving the name of the file.

Examples

```
LIST
lis *.MSH
```

A.7 WRITE - to provide monitoring of a session

Syntax

```
WRIte STAte=<choice> [FIle=<string>]
      [PROmpt=<choice>]
```

Description

Redirects the command decoder echo output to the file specified by the FILE parameter. The information flow is controlled by the STATE parameter. This command can enable the constructions of command files to drive the program in a background mode.

The echoing of the command prompt can be controlled using the PROMPT parameter.

Parameters

STATE	required *choice* Controls the flow of information to the monitoring file It has values NO, OFF or CLOSE. ON switches on monitoring. OFF suspends it but does not close the file and CLOSE ends monitoring and closes the file.
FILE	retained *string* : initial = MONITOR Output file name to receive the monitoring stream.
PROMPT	reset *choice* : initial = OFF Allows you to select whether the command prompt is echoed in the monitoring file. It has values ON or OFF.

Examples

```
WRITE STATE=ON FILE=MONITOR PROMPT=OFF
wri on junk on
```

A.8 READ - to specify a command input file

Syntax

```
REad FIle=<string> [ECHO=<choice>]
```

Description

Redirects the command decoder to take its input from a file specified by the FILE parameter. However, if FILE is given as TERMINAL, input returns to the standard input stream.

The echoing of the commands being read by the decoder can be controlled using the ECHO parameter.

Parameters

FILE	required *string* Input file name containing program commands.
ECHO	reset *choice* : initial = OFF Echo control option. Values can be ON or OFF.

Examples

In this example a sequence of commands are read from the file NAIL and ECHOed to the standard output device:

```
Everest: read nail echo=on
Everest: GEO * >DATA>NAIL.GEO
Everest: MES * >DATA>OXNAIL.MSH
Everest: DOP * >DATA>NAIL.DOP
Everest: RES NAIL.OUT R
Everest: PHY NAIL.PHY
Everest: BIAS LEFT-CNT 0
Everest:
```

A.9 SYNTAX - to provide the syntax of a command

Syntax

```
SYNtax [COMmand=<string>]
```

Description

Displays the formal syntax of all the currently defined commands. If the syntax of a specific command name is required then that name is given as a parameter to the command.

Parameters

COMMAND	retained *string* : initial = ALL Specifies the commands name for which the syntax is required. If the syntax of all the currently defined commands is required, then the special command name ALL should be used.

Examples

The following example obtains the syntax of all the commands in the DOCUMENT program.

```
Doc: syntax

The commands currently defined are:

SYNtax [COMmand=<string>]
Help [KEY=<string>] [OPTion=<string>]
```

```
FILe INput=<string>, OUTput=<string>
PROcess
Quit
TItle TEXT=<string>
AUthor TEXT=<string>
Date TEXT=<string>
OPtions [SORT=<choice>] [CONtents=<choice>]
        [RUNoff=<choice>] [FRont_page=<choice>]

 For further information type: HELP <command name> [<option>],
 where <option> is BRIEF or FULL

 Doc:
```

A.10 HELP - to access HELP system

Syntax

```
Help [KEY=<string>] [OPTion=<choice>]
```

Description

Accesses to the inbuilt HELP system within the command decoder. HELP is one of the internal commands of the command processor and has a companion command SYNTAX.

HELP has two parameters allowing the selection of help on a specific command and the level of help required (SUMMARY, BRIEF, FULL and SYNTAX). If no command name is given summary help is given on all the commands currently defined.

If an ambiguous or invalid command name is given a warning or error message is given.

BRIEF help gives information on the purpose, syntax and the current state of the selected command. A table of command keywords, their type, status and current value (if applicable) is printed.

When the FULL option is used the Help System uses the inbuilt free text retrieval system to access the help data base. This allows the display of the full command description and the searching for specific keywords. This option is not supported in the current release.

Parameters

KEY	reset *string* : initial = Either the global command name SUMMARY, or the specific command name on which help is sought.
OPTION	reset *choice* : initial = BRIEF The level of help required. This can be SUMMARY, BRIEF, FULL or SYNTAX.

Examples

```
Everest: help output

Name     : OUTPUT

Purpose  : to specify results file
Syntax   : OUTput FILE=<string> [REPLACE=<choice>]


Keyword     Type             Status        Current Value
---------------------------------------------------------------

FILE        string           required       undefined
REPLACE     choice           reset          replace,NOREPLACE
```

B Geometry Command Reference

B.1	CONTACT	to define a contact
B.2	GEOM	to process the geometry input
B.3	LINE	to define a line
B.4	NEUTRAL	to specify the output file for the geometry
B.5	POINT	to specify a point
B.6	QUIT	to leave the geometry input sections without data verification
B.7	RECAP	to review the geometric data entered
B.8	SURFACE	to specify a surface
B.9	VOLUME	to specify a volume
B.10	WINDOW	to specify a window

B.1 CONTACT - to define a contact

Syntax

```
CONtact NAME=<string> SURFACES=<string_list>
```

Description

Defines the geometric properties of a contact. The contact is defined in terms of surfaces and windows which the user defines also.

Parameters

NAME	Required *string* A *string* giving the name of the contact as used in the Solution Module.
SURFACES	Required *string_list* A *string_list* giving the names of the surfaces and windows which define the contact. It is not necessary for both contacts and windows to be used in the definition.

Examples

```
CONTACT NAME=ANODE, SURFACES=(SURF1,SURF5,WIND3)
CON CATHODE(SURF4)
```

B.2 GEOM - to process the geometry input

Syntax

```
GEom
```

Description

Ends your geometric input. The data is then checked for consistency by the geometry code. Any errors detected are announced.

Parameters

This command has no parameters.

Examples

```
GEOM
GE
```

B.3 LINE - to define a line

Syntax

```
LIne NAME=<string> POINTS=<string\_list>
```

Description

Defines a line which is used in the definition of either a surface or a window. The line, which must be straight, is defined by a collection of two or more points. The ability to include more than two points permits greater flexibility in the definition of mesh refinements on a given line.

Parameters

NAME	Required *string* A *string* giving the name of the line as used in the definition of a surface or a window.
POINTS	Required *string_list* A *string_list* giving the names of the points which define the line. They must be given in the order in which they are met travelling from one end of the line to the other; the direction itself is irrelevant.

Examples

```
LINE NAME=L12 , POINTS=( P1 , P2 )
LIN L45 (P17 P19 P22 P1)
```

B.4 NEUTRAL - to specify the output file for the geometry

Syntax

```
NEutral NAME=<string>
```

Description

Specifies the name of the output file for the geometric data. This file will contain the geometric records of the neutral file.

Parameters

NAME	Required *string* A *string* giving the name of the file to which the geometric data is to be written.

Examples

```
NEUTRAL NAME=FIASCO
NE MOS_TEST1
```

B.5 POINT - to specify a point

Syntax

```
POint NAME=<string> X=<real> Y=<real> Z=<real>
```

Description

Defines a point which is used in the definition of a line.

Parameters

NAME	Required *string* A *string* giving the name of the point as used in definition of a line.
X	Required *real* A *real* giving the x co-ordinate of the point. The unit is μm.
Y	Required *real* A *real* giving the y co-ordinate of the point. The unit is μm.
Z	Required *real* A *real* giving the z co-ordinate of the point. The unit is μm.

Examples

```
POINT NAME=P12 , X=12.0 , Y=14.2 , Z=0.0
PO P4 10 5 5
```

B.6 QUIT - to leave the geometry input section without data verification

Syntax

```
QUit
```

Description

Allows you to leave the geometry input section without verifying the data. This sets a flag in the program to prevent you from submitting this model to the mesh generator for meshing.

Examples

```
QUIT
QU
```

B.7 RECAP - to review the geometric data entered

Syntax

```
RECAP
```

Description

Enables you to see the commands entered to-date when you are entering data interactively.

Parameters

This command has no parameters.

Examples

```
RECAP
RE
```

B.8 SURFACE - to specify a surface

Syntax

```
SUrface NAME=<string> LINES=<string\_list>
```

Description

Defines a surface which is used in the definition of a volume. The surface may also form part of the definition of a contact. Currently the surface must be rectangular and aligned with two of the major axes. In the second phase of the software development the alignment restriction will be lifted. Later, the definition of irregular, simply-connected planar, surfaces will be permitted.

Parameters

NAME	Required *string* A *string* giving the name of the surface as used in definition of a volume.
LINES	Required *string_list* A *string_list* giving the names of the points which define the line. They need not be in any particular order but each must be defined by a LINE command (cf. B.3).

Examples

```
SURFACE NAME=INTERFACE , ( L12 , L34 , L9110 , L45 )
SU S47 ( L12 , L23 , L34 , L45 , L51 )
```

B.9 VOLUME - to specify a volume

Syntax

```
VOlume NAME=<string> SURFACES=<string\_list>,
       MATERIAL=<string>
```

Description

Defines a volume which forms part or all of a semiconductor device. In the current release of the code the volumes must be regular hexahedra whose sides are aligned with the major axes. In the next release of the software non-rectilinear hexahedra will be permitted. Eventually the definition of general three-dimensional shapes will be permitted.

Parameters

NAME	Required *string*. A *string* giving the name of the volume as used in definition of a device structure.
LINES	Required *string_list*. A *string_list* giving the names of the surfaces which define the volume. They need not be in any particular order but each must be defined by a SURFACE command (cf. B.8).
MATERIAL	Required *string*. A *string* giving the name of the material of which the volume is constituted. The recognised material types are silicon, oxide, nitride and air. This list may be augmented in the future.

Examples

```
VOLUME NAME=P_REGION,SURFACES=(S12,S47,JUNCTION,S1,S3,S5),
       MATERIAL=SILICON
VO V4 ( S1 , S4 , S19 , S22 , S23 , S34 , S2 ) SI
```

B.10 WINDOW - to specify a window

Syntax

```
WIndow NAME=<string> LINES=<string\_list> SURFACE=<string>
       [LSYMM=<string\_list>]
```

Description

Defines a rectangular window on the surface of the device; this may be considered as a sub-surface. The window may be used in the definition of a contact or may be used as a window through which impurities are injected into the device in the calculation of the impurity profile. You may use the LSYMM option to specify any lines of symmetry required in the definition of the impurity profile. In this release of the code the window must be aligned with two of the major axes.

Parameters

NAME	Required *string*. A *string* giving the name of the window as used in the definition of a contact or in the generation of an impurity profile.
LINES	Required *string_list*. A *string_list* giving the names of the lines used to define the window. They need not be in any particular order but each of these must be defined by a LINE command (cf. B.3).
SURFACE	Required *string*. A *string* giving the name of the surface of which this is a sub-surface. This surface must be defined by a SURFACE command (cf. B.8).
LSYMM	Optional *string_list*. A *string_list* giving the names of the defining lines which coincide with lines of doping symmetry. If a window definition has no such lines it is omitted from the definition of the window. It is reset to null after use.

Examples

```
WINDOW NAME=ANODE, LINES=(L12, L23, L34, L41), SURF2,
       LSYMM=(L12, L41)
WI GAUSS1 ( L47 , L75 , L52 , L24 ) S2
```

C Standard Models Command Reference

D.1	BIPOLAR	to define a bipolar polysilicon emitter by giving its dimensions
D.2	MOSFET	to define a MOSFET by giving its dimensions
D.3	NEUTRAL	to specify the output file for the geometry or mesh
D.4	ONE_DIODE	to define a one-dimensional diode by giving its dimensions
D.5	QUIT	to leave the standard input section without data verification
D.6	STANDARD	to process the standard model