PISCES-IIB

Supplementary report

Mark R. Pinto

Conor S. Rafferty

Hal R. Yeager

Robert W. Dutton

Stanford Electronics Laboratories
Department of Electrical Engineering
Stanford University, Stanford Ca 94305.

Chapter 1

Introduction

The following is a description of the PISCES-IIB semiconductor device modeling program. This document is intended to be a supplement to the report on the earlier version of this program, PISCES-IIA [1].

PISCES-IIB contains a number of enhancements over version II-A. The major additions include a second-order time discretization and automatic time-step selection algorithm, lumped resistive and capacitive boundary conditions, current boundary conditions, distributed contact resistance, small-signal analysis, Fermi-Dirac statistics (including incomplete ionization) and a new grid generation program (written in C). Many other enhancements and improvements have been made and are either explicitly described in this document or are simply included in the revised description of the input language.

This report begins in chapter 2 with a brief description of a number of the new features as mentioned above. In chapter 3, some observations will be made with regard to general simulation issues. Chapter 4 is a guide to the new grid generation system, while chapter 5 contains simple, reproducible examples incorporating some of the new analytical capability. Finally the appendix contains a complete description of the revised input language.

References

[1] Mark R. Pinto, Conor S. Rafferty and Robert W. Dutton, ‘‘PISCES-II - Poisson and Continuity Equation Solver,’’ Stanford Electronics Laboratory Technical Report, Stanford University, September 1984.

Chapter 2

Major Enhancements

0.1. Transient Device Simulation

PISCES-II assumes that the electrical behavior of semiconductor devices is governed by Poisson's equation

$$\varepsilon \nabla^2 \psi = -q\left(p - n + N_D^+ - N_A^-\right) - \rho_F \tag{2.1}$$

and the continuity equations for electrons and holes

$$\frac{\partial n}{\partial t} = \frac{1}{q} \nabla \cdot J_n - U_n = F_n(\psi, n, p) \tag{2.2}$$

$$\frac{\partial p}{\partial t} = -\frac{1}{q} \nabla \cdot J_p - U_p = F_p(\psi, n, p) \tag{2.3}$$

In this section, we are concerned only with the time-dependent discretization of (2.1)-(2.3). See [1] and section 2.4 of this report for a description of the auxiliary physical considerations and [1-3] for the details of the spatial discretization for these coupled partial differential equations.

Because of the extremely stiff nature of (2.1)-(2.3), strong stability requirements are placed on any proposed transient integration scheme. Additionally, it is most convenient to use one-step integration methods so that only the solution at the most recent time-step is required. Most device simulation codes (including PISCES-IIA) have therefore made use of a simple first-order (implicit) backward difference formula [4,5], i.e. (2.2) and (2.3) are discretized as

$$\frac{n_k - n_{k-1}}{\Delta t_k} = F_n(\psi_k, n_k, p_k) = F_n(k) \tag{2.4}$$

$$\frac{p_k - p_{k-1}}{\Delta t_k} = F_p(\psi_k, n_k, p_k) = F_p(k) \tag{2.5}$$

where $\Delta t_k = t_k - t_{k-1}$ and ψ_k denotes the potential at time t_k, etc.. This scheme (BDF1) is a one-step method and is known to be both A- and L-stable [4,6] but suffers from a large local truncation error (LTE) which is proportional to the size of the time-steps taken, i.e. the error at each time-step k is $O(\Delta t_k)$.

Because it is a one-step, second-order ($O(\Delta t_k^2)$), A-stable (implicit) integration scheme, the trapezoidal rule (TR) [4] is a very attractive alternative to BDF1 for a number of problems involving the solution of differential equations. Applying TR to (2.2) and (2.3) we obtain

$$\frac{n_k - n_{k-1}}{\Delta t_k} = \frac{1}{2}\left[F_n(k) + F_n(k-1)\right] \tag{2.6}$$

$$\frac{p_k - p_{k-1}}{\Delta t_k} = \frac{1}{2}\left[F_p(k) + F_p(k-1)\right] \tag{2.7}$$

Unfortunately, apart from being slightly more difficult to implement, TR is not in general L-stable so that numerical "ringing" may be observed unless very small time-steps (much smaller than dictated by LTE considerations) are chosen. In fact, this behavior is indeed the case with the semiconductor device equations, as noted first in [5], due to the extreme stiffness of the problem as mentioned above.

An alternative second-order method might be a second-order backward difference formula (BDF2) [4], i.e.

$$\frac{\frac{2-\gamma}{1-\gamma}\, n_k - \frac{1}{\gamma(1-\gamma)}\, n_{k-1} + \frac{1-\gamma}{\gamma}\, n_{k-2}}{t_k - t_{k-2}} = F_n(k) \qquad (2.8)$$

$$\frac{\frac{2-\gamma}{1-\gamma}\, p_k - \frac{1}{\gamma(1-\gamma)}\, p_{k-1} + \frac{1-\gamma}{\gamma}\, p_{k-2}}{t_k - t_{k-2}} = F_p(k) \qquad (2.9)$$

where

$$\gamma = \frac{t_{k-1} - t_{k-2}}{t_k - t_{k-2}} \qquad (2.10)$$

Like BDF1, BDF2 is both A- and L-stable, but BDF2 requires two previous solutions at times $k-1$ and $k-2$ (i.e. it is not a one step method) so that restarting is generally more difficult. Additionally, although both are second order methods, BDF2 has a higher LTE than TR due to the associated proportionality factor.

Recently, Bank, et. al. [7] proposed a composite TR-BDF2 method which is one-step, second-order and both A- and L-stable. The basic algorithm is to divide each time interval Δt_k into two parts - $\gamma\Delta t_k$ and $(1-\gamma)\Delta t_k$; TR (2.6, 2.7) is then used in the first sub-interval, and BDF2 (2.8, 2.9) is used in the second. Additionally, if γ is chosen properly it can be shown that the Jacobian need not be re-factored for the TR step, substantially reducing the cpu time per time-step.

In PISCES-IIB, the TR-BDF2 composite integration scheme of Bank, et. al. has been implemented. The method has been found to be strictly second-order and stable for every device and bias condition tested. In order to simplify the simulation procedure, an automatic time-step selection algorithm based on the LTE of the TR-BDF2 method has been included. Specifically, at each time point t_k, the LTE can be estimated by the divided-difference formula [7]

$$LTE = C(\Delta t_k)^3 \frac{d^3 F_{n,p}}{dt^3}$$

$$\approx 2C(\Delta t_k)\left[\gamma^{-1}F_{n,p}(k) - \gamma^{-1}(1-\gamma)^{-1}F_{n,p}(k-1) + (1-\gamma)^{-1}F_{n,p}(k-2)\right] \qquad (2.11)$$

where C is a pre-defined constant. The new time-step Δt_{k+1} is then selected by

$$\Delta t_{k+1} = \Delta t_k \left[\frac{tol}{||\tau_i||}\right]^{1/3} \qquad (2.12)$$

where τ_i is the relative LTE for each node i and tol is a specified LTE tolerance. Special care is taken not to let the time step grow by more than a certain factor (usually 2). On some occasions (usually with a very large specified LTE tolerance), PISCES will decide to go back to the solution for previous time and re-solve with a smaller time-step.

In PISCES-IIB, all the user must specify is an initial time-step as well as an input bias condition (voltage/current, ramp/step). All the numerical parameters have defaults and most are accessible. Time-steps are stored for each solution so that a transient simulation can be continued from a saved solution file without specifying a new initial time-step. Note that the intermediate solution of the composite step is not printed or saved. See the descriptions of the METHOD and SOLVE cards in the appendix and the example in chapter 5 for more details.

One final comment on general transient device simulation : if a voltage step (or ramp, which is just small incremental steps) is applied directly to a semiconductor contact (i.e., not through a resistor - see next section), the resultant terminal currents will include an impulse response term (delta function) due to the infinitely fast change in voltage and the intrinsic capacitance of the device, which depends on the size of the simulation space and dielectric constants. It may be advisable to try and resolve this component if a voltage input is being applied directly.

0.2. New Boundary Conditions

PISCES-IIA included the basically four types of boundary conditions : ohmic contacts, Schottky contacts, insulator contacts and Neumann (reflective) boundaries (see section 2.4 of [1]). Continued use of the program for a number of specific applications has led to the implementation in PISCES-IIB of additional (supplemental) boundary conditions. First, in order to save grid space, it would be quite beneficial to include lumped resistances and/or capacitances between applied biases and semiconductor device contacts. Second, for devices such as SCR's where current is a multi-valued function of applied voltage, a current boundary condition would be quite useful. Finally, to account for the finite resistivity of contacts to the semiconductor, a true distributed contact resistance might be desirable. These three auxiliary boundary conditions are described in detail below.

0.2.1. Lumped elements

The need for lumped elements was basically motivated by two desires: to cut down on the number of grid points used to discretize some device structures (i.e., to save cpu time) and to handle transient simulation in a more realistic manner (see comments in section 1 of this chapter). A typical example of a case where a lumped resistance is useful is shown in Figure 1. The p-tub (base) contact in this CMOS cross-section might be tens or hundreds of microns away from the active area of the embedded vertical npn bipolar transistor, which may only be $10 - 20\mu$m on a side. If the whole structure were to be simulated, a tremendous number of grid points, probably more than half, would be wasted in order to account for a purely resistive region of the device. Because cpu time is generally a super-linear function of the number of grid points for all numerical methods of interest [2], including such regions explicitly can be quite expensive. Another example might be the substrate of a MOSFET, where we most certainly would not want to include grid points all the way to the back side of a wafer! In both of these cases, a simple lumped resistance might be substituted. Of course similar arguments hold true for capacitance.

Figure 2 shows a schematic of how these boundary conditions might be implemented (momentarily ignore the current source - it will be treated later). An extra unknown, the voltage on the semiconductor contact (ϕ), must be included and is defined by a Kirchoff equation:

$$\frac{V_{app} - \phi}{R} + C\,\frac{d(V_{app} - \phi)}{dt} - \sum_{i=1}^{N_b}(J_n + J_p + J_{disp})_i = 0 \tag{2.13}$$

where N_b is the number of boundary grid points associated with the electrode of interest. Note that this auxiliary equation, due to the currents inside the summation, has dependencies on the values of potential and carrier concentrations (represented as x in the figure) at the nodes on the electrode as well as all nodes directly adjacent to the electrode. Note also that the temporal term associated with the capacitor must be discretized in a manner consistent with the device equations (see section 1 above). It is important to remember that in contrast to the distributed resistance to be described below, a lumped element contact will have a single voltage (or potential - adjusted for possible doping non-uniformities) associated with the entire electrode.

Within PISCES-IIB, lumped resistances and capacitances are specified on the CONTACT card (see appendix). Note that because an auxiliary equation must be added to the system, the symbolic matrix factorization (SYMBOL card) must follow all CONTACT cards in an input deck. The values of the resistances should be in units of $\Omega - \mu$m while capacitance should have the units F/μm. Remember that capacitance increases with device width (into the z plane) while conversely, resistance decreases. Except for the case of extremely large resistances, where the arrangement becomes similar to a pure current source (see below), no convergence degradation has been observed for a lumped element boundary in comparison to a simple ohmic contact. Further, as mentioned in section 1 above, transient simulation becomes easier and is more well-defined.

One final comment: one should always make use of the simulator as much as possible to help calculate any resistance (or capacitance) components that might be included as lumped elements. For instance in the case of Figure 1, one could simulate just the p-tub with ohmic contacts at either end. From the plot of terminal current (in A/μm) versus voltage, the resistance can be directly extracted from the slope. Be very careful however to account for any three dimensional effects (e.g., current spreading) before using such a resistance value in further simulations.

0.2.2. Current boundary conditions

One of the first applications of PISCES-II was in the analysis of CMOS latch-up triggering [8]. Shown in Figure 3 are simulated (by PISCES-II) ‘‘low’’ current IV characteristics of two different embedded SCR structures found in a typical VLSI CMOS process as used in [8]. Note that as mentioned previously, the terminal current is a multi-valued function of the applied voltage. This condition implies that for some voltage boundary conditions, a numerical procedure may end up - depending on the initial guess - with a solution in either of two (actually three - the third, higher current state is not shown) distinct and stable states. Furthermore, the condition of primary interest is the point at which $dV/dI = 0$, known as the trigger point, which is exceedingly difficult to obtain with a simple voltage boundary condition. Additionally, it is nearly impossible to compute any solutions in the negative resistance regime with voltage inputs.

A possible alternative would be to define a current boundary condition, since voltage can be thought of as a single-valued function of the terminal current (just turn Figure 3 on its side). Such a boundary condition has been implemented within PISCES-IIB, again as an auxiliary equation with an additional unknown boundary potential. Like the lumped R/C case, a Kirchoff equation is written at the electrode :

$$J_{source} - \sum_{i=1}^{N_b} (J_n + J_p + J_{disp})_i = 0 \tag{2.14}$$

Note that unlike the lumped R/C case, J_{source} is constant and has no dependence on the boundary potential ϕ (the ϕ dependence is buried in the summation). Because of this weaker dependence on ϕ, the convergence of the non-linear Newton iteration is affected, but not always for the worse. Table I shows the number of Newton loops needed for various bias conditions using both voltage and current boundary conditions (with ‘‘previous’’ solution used as initial guess).

Table I - Voltage vs. Current BC

V_F	I (Amps/μm)	Voltage BC	Current BC
0	$0.00\text{x}10^{0}$	8	-
0.1v	$8.02\text{x}10^{-15}$	5	6
0.2v	$9.50\text{x}10^{-14}$	5	7
0.3v	$1.96\text{x}10^{-12}$	5	8
0.4v	$6.98\text{x}10^{-11}$	5	9
0.5v	$2.92\text{x}10^{-9}$	5	9
0.6v	$8.07\text{x}10^{-8}$	6	8
0.7v	$9.11\text{x}10^{-7}$	7	6
0.8v	$6.73\text{x}10^{-6}$	6	5
0.9v	$3.45\text{x}10^{-5}$	6	5
1.0v	$7.48\text{x}10^{-5}$	5	5
1.1v	$1.01\text{x}10^{-4}$	5	4
1.2v	$1.23\text{x}10^{-4}$	5	4
1.3v	$1.43\text{x}10^{-4}$	4	4

Note that for regions where a small dI/dV is small, the voltage boundary condition is definitely preferable. However, as evidenced by its slight advantage as the diode turns on, the current boundary condition may be preferable for operating regimes where dI/dV is large. One note here: it is not uncommon for the negative resistance regime of an SCR to have a slope dI/dV very close to 0. Such behavior should be considered when using a current source to trace out an entire IV curve; i.e., it might be preferable to switch back to a voltage source after passing the trigger point.

0.2.3. Distributed contact resistance

As mentioned above, for a contact with a lumped element or for a simple voltage boundary condition, a single potential is associated with the entire electrode. However, because contact materials have finite resistivities, the electrostatic potential is not truly uniform along the metal-semiconductor interface. To account for this effect, a distributed contact resistance can be associated with any electrode in a PISCES-IIB simulation.

The implementation of distributed contact resistance is as follows : PISCES-IIB internally places a resistance R_i at each node i associated with the contact of interest. The value of each R_i is computed from the contact resistance, ρ_c (in $\Omega - \text{cm}^2$) as

$$R_i = \rho_c d_i \tag{2.15}$$

where d_i is the length of the contact associated with node i. An auxiliary equation is then added for each electrode node, unlike the lumped element case where a single equation is added for the entire electrode. So for every node i that is part of the contact,

$$\frac{V_{app} - \left(\psi_i \pm \frac{kT}{q} \log \frac{N}{n_i} \right)}{R_i} - (J_n + J_p + J_{disp})_i = 0 \tag{2.16}$$

Note however that the extra equations added to the system are strictly local in nature. Only the current at node i is included in (2.16) as opposed to the summation over the contact in (2.13), so that there is no direct coupling between non-adjacent electrode nodes and neighbors. As such it is actually somewhat simpler to implement numerically than the lumped element or current boundary conditions.

Figure 4 shows a typical example as simulated by PISCES-IIB. The device under consideration (Figure 4b) is a uniformly doped slab (4μm by 0.2μm) of n-type silicon with a sheet resistance of 25Ω/sq (note that the figures are not drawn to scale). Figure 4b,c shows the current vectors for 0 and $10^{-6}\Omega - \mathrm{cm}^2$ respectively with V=0.2v. Note the way the current crowds into the nearest corner of the electrode on the right in Figure 4a, whereas in Figure 4b, the current is evenly distributed along the contact. As evidenced by the magnitude of the arrows, the current with 0 contact resistance is much larger. Figure 4c shows the IV characteristics for both structures. The slope of the line corresponding to the device with contact resistance is equivalent to a total resistance of $\approx 116\Omega - \mu\mathrm{m}$ which agrees exactly with the analytical solution for this case.

0.3. Small-Signal Analysis

PISCES-IIA included two basic types of electrical analysis: DC steady state and transient analysis. In PISCES-IIB, in addition to enhancing the transient analysis (see section 1 of this chapter), AC small-signal analysis has been added. Specifically, starting from a DC bias condition, an input of given amplitude and frequency can be applied to a device structure from which sinusoidal terminal currents and voltages are calculated. Then using the relationship

$$\tilde{Y}_{ij} = G_{ij} + j\omega C_{ij} = \frac{\tilde{I}_i}{\tilde{V}_j} \quad , \quad \tilde{V}_k = 0 \;\; k \neq j \tag{2.17}$$

the frequency dependent admittance matrix, and hence capacitance, can be calculated. Note also that by varying the frequency and examining the various device admittances, f_T can be directly determined.

PISCES-IIA has been successfully applied to the calculation of bias dependent gate capacitances for small geometry MOSFETs [9]. By solving for a bias point (V_G, V_D, V_S, V_B) and then solving again for (V_G, $V_D + \Delta V$, V_S, V_B), one can obtain very accurate estimates of capacitance by then integrating the charge on the gate electrode at both bias points, i.e.

$$C_{ij} = C_{GD} = \frac{dQ_G}{dV_D} \approx \frac{\Delta Q_G}{\Delta V} \tag{2.18}$$

Of course, this procedure can be repeated to obtain C_{GS} and C_{GB} by applying the potential increment to the source and bulk respectively. Unfortunately, as pointed out by Laux [10], this technique of charge partitioning is only rigorously correct for cases where the current into electrode i (to calculate C_{ij}) is strictly displacement current, i.e. insulator contacts like MOS gates. Therefore, capacitances such as C_{DG} and C_{SG} cannot be accurately estimated by this method. Additionally, the capacitances obtained via this method are strictly quasi-static (low frequency) values.

Using sinusoidal steady-state analysis, both of the drawbacks associated with the charge partitioning approach are eliminated. In PISCES-IIB, the approach of Laux [10] has been followed. An ac sinusoidal voltage bias is applied to an electrode i such that

$$V_i = V_{i0} + \tilde{V}_i e^{j\omega t} \tag{2.19}$$

where V_{i0} is the existing dc bias, $\tilde{V}_i$ is the magnitude of the ac sinusoidal bias and V_i is the actual bias (sum) to be simulated. Rearranging the basic partial differential equations (2.1)-(2.3), one obtains

$$F_\psi(\psi, n, p) = \varepsilon \nabla^2 \psi + q\left(p - n + N_D^+ - N_A^-\right) + \rho_F = 0 \tag{2.20}$$

$$F_n(\psi, n, p) = \frac{1}{q} \nabla \cdot J_n - U_n = \frac{\partial n}{\partial t} \tag{2.21}$$

$$F_p(\psi, n, p) = -\frac{1}{q} \nabla \cdot J_p - U_p = \frac{\partial p}{\partial t} \tag{2.22}$$

The ac solution to (2.20)-(2.22) can be written as

$$\psi_i = \psi_{i0} + \tilde{\psi}_i e^{j\omega t} \tag{2.23}$$

$$n_i = n_{i0} + \tilde{n}_i e^{j\omega t} \tag{2.24}$$

$$p_i = p_{i0} + \tilde{p}_i e^{j\omega t} \tag{2.25}$$

where ψ_{i0}, n_{i0} and p_{i0} are the dc potential and carrier concentrations at node i while $\tilde{\psi}_i$, $\tilde{n}_i$ and $\tilde{p}_i$ are the respective ac values, which in general are complex. By substituting (2.23)-(2.25) back into (2.20)-(2.22) and expanding as a Taylor series to first-order only (the small-signal approximation), we obtain non-linear equations of the form

$$F(\psi, n, p) = F(\psi_0, n_0, p_0) + \frac{dF}{d\psi}(\tilde{\psi} e^{j\omega t}) + \frac{dF}{dn}(\tilde{n} e^{j\omega t}) + \frac{dF}{dp}(\tilde{p} e^{j\omega t}) \tag{2.26}$$

for each of the three PDE's. If we have already have computed a valid dc solution at the desired dc bias, then $F(\psi_0, n_0, p_0) = 0$ so the following linear system is obtained:

$$\begin{bmatrix} \dfrac{dF_\psi}{d\psi} & \dfrac{dF_\psi}{dn} & \dfrac{dF_\psi}{dp} \\ \dfrac{dF_n}{d\psi} & \dfrac{dF_n}{dn} + D_1 & \dfrac{dF_n}{dp} \\ \dfrac{dF_p}{d\psi} & \dfrac{dF_p}{dn} & \dfrac{dF_p}{dp} + D_1 \end{bmatrix} \begin{bmatrix} \tilde{\psi} \\ \tilde{n} \\ \tilde{p} \end{bmatrix} = b_1 \tag{2.27}$$

where each of the $dF/d\psi$, dF/dn and dF/dp are n-by-n matrices which form the dc Jacobian, D_1 is an n-by-n matrix with $-j\omega$ on the diagonal and 0 off-diagonal (resulting from the expansion of the time-dependent portion of each continuity equation) and b_1 is a vector of length 3n which contains the ac input voltage boundary conditions.

Splitting the system (2.27) into real and imaginary parts, one obtains

$$\begin{bmatrix} J & -D_2 \\ D_2 & J \end{bmatrix} \begin{bmatrix} X_R \\ X_I \end{bmatrix} = b_2 \tag{2.28}$$

where J is the 3n-by-3n dc Jacobian, D_2 is a 3n-by-3n diagonal matrix (related to D_1 above), b_2 is a permuted version of b_1 above and X_R and X_I refer to the real and imaginary ac solution vectors, each 3n in length. The system (2.28) is twice the size of the dc system. However, because we have already factored J if the full Newton method was used (required by PISCES for this analysis), an iterative method like SOR can be used on (2.28) to obtain the ac solution. Convergence is generally very good for low to moderate frequencies; however as the frequency reaches rather high values ($\approx f_T/10$), more iterations are necessary. For frequencies above $f_T/10$, a relaxation parameter with a value less than 1 may have to be used. For exceedingly high frequencies, SOR will not converge at all. See Laux [10] for more details.

PISCES-IIB performs ac analysis as a post-processing step after a dc solution. Specific details are given in the documentation in the appendix regarding the SOLVE card, and an example of ac analysis is included in the example in chapter 5.

0.4. Enhanced Physics

PISCES-IIA has a very comprehensive set of models to accurately describe the actual physical mechanisms associated with semiconductor device behavior. However, as mentioned in chapter 2 of the first technical report on PISCES-II [1], neither Fermi-Dirac statistics nor incomplete ionization of impurities were included in version II-A. They have been included in PISCES-IIB and are described below. Additionally, surface recombination at insulator-semiconductor interfaces and barrier lowering at Schottky contacts have also now been included.

We fully realize that there are a number of other important physical effects for which we have not explicitly included models. Therefore, included elsewhere in this document is a brief discussion on how one might implement new models into the code.

0.4.1. Fermi-Dirac Statistics

The formulation of Fermi-Dirac statistics follows that developed by Yu [11] where the carrier concentrations are given by equation (2.29):

$$n = N_C \gamma_n e^{\eta_n} \tag{2.29a}$$

$$p = N_V \gamma_p e^{\eta_p} \tag{2.29b}$$

where η_n, η_p, γ_n and γ_p are defined in terms of the quasi-fermi levels (E_{Fn} & E_{Fp}) and band energies (E_C & E_V):

$$\eta_n = \frac{E_{Fn} - E_C}{kT} \tag{2.30a}$$

$$\eta_p = \frac{E_V - E_{Fp}}{kT} \tag{2.30b}$$

$$\gamma_n = \frac{F_{1/2}(\eta_n)}{\exp(\eta_n)} \tag{2.30c}$$

$$\gamma_p = \frac{F_{1/2}(\eta_p)}{\exp(\eta_p)} \tag{2.30d}$$

In effect, equations (2.29) and (2.30) introduce a form one ($\exp(\eta)/\exp(\eta)$) into the Fermi-Dirac statistic. The identification of the exponential term in (2.29) facilitates the standard Scharfetter-Gummel discretization of the continuity equations. The actual code implementation requires both the Fermi-Dirac function $F_{1/2}(\cdot)$ and its inverse $F_{1/2}{}^{-1}(\cdot)$. The inverse function is calculated from the Joyce-Dixon approximation [12] given by equation (2.31):

$$F_{1/2}{}^{-1}(X_S) = \eta_S = \ln(X_S) + aX_S + b(X_S)^2 + c(X_S)^3 + d(X_S)^4 \tag{2.31}$$

where η_S represents either η_n or η_p and X_S denotes either n/N_C or p/N_V. The constants a, b, c, and d can be found in either [12] or [11]. For values of $X_S > 8.0$, the asymptotic expansion for the Fermi-dirac inverse is used:

$$\eta_S = \left(\left(\frac{3\sqrt{\pi}}{4} X_S \right)^{4/3} - \frac{\pi^2}{6} \right)^{1/2} \tag{2.32}$$

In either case, the Fermi-Dirac function is calculated from its inverse through a straight forward inversion process.

Fermi-Dirac statistics can be selected by setting the parameter ‘FERMIDIRAC = true’ on the MODELS card. It is strongly recommended that incomplete ionization of donors also be included for accurate simulations (see next section).

0.4.2. Incomplete Ionization

PISCES-IIB now associates an incomplete ionization to p-type and n-type doping:

$$N_D^{\,+} = \frac{N_D}{1 + g_D \exp\left(\frac{E_{Fn} - E_D}{kT}\right)} \tag{2.33a}$$

$$N_A^{\,-} = \frac{N_A}{1 + g_A \exp\left(\frac{E_A - E_{Fp}}{kT}\right)} \tag{2.33b}$$

where N_D and N_A are net compensated n-type and p-type doping respectively (ie. if $N_{total} \equiv (N_{D,total} - N_{A,total}) > 0$, then $N_D := |N_{total}|$ and $N_A := 0$ else $N_D := 0$ and $N_A := |N_{total}|$).

Incomplete Ionization can be selected by setting the parameter ‘INCOMPLETE = true’ on the MODELS card. The degeneracy factors g_D and g_A are located on the MATERIAL card and specified as ‘GCB’ and ‘GVB’ respectively. The donor ionization energies are also located on the MATERIAL card and are referenced to the corresponding energy band:

$$‘EDB’ = E_C - E_D \tag{2.34a}$$

$$‘EAB’ = E_A - E_V \tag{2.34b}$$

‘EDB’ and ‘EAB’ are, in general, positive quantities.

0.4.3. Low Temperature Simulations

In conjunction with the Fermi-Dirac statistics and donor freeze-out, the PISCES-IIB program has been restructured to allow low temperature simulations. In general, simulations can be done down to $50°K$ without loss of quadratic convergence. Below this temperature, the carrier and ionization statistics develop sharp transitions which induces severe damping, hence loss of quadratic convergence, in the non-linear newton step. Since several iterations are required below 50°K, the ‘ITLIMIT’ parameter on the METHODS card should be increased.

Due to the limited exponent range on some machines, trouble in calculating the quasi-fermi level of minority carriers may be encountered. As the temperature decrease, the intrinsic carrier concentration n_i also decreases (for example $n_i \approx 10^{-10}$ at 100°K for silicon). In the quasi-neutral regions, the minority carrier concentration can easily underflow. PISCES-II has always handled such situations by setting those concentrations to zero. This, however, does not allow an accurate post-calculation of the minority carrier quasi-fermi levels. To compensate the quasi-fermi level calculations, the majority carrier concentration and the relation $np = n_i^2$ are used to deduce the minority carrier concentrations should they underflow. Despite these efforts, spurious ‘glitches’ are occasionally observed at low temperatures in minority quasi-fermi levels. The current calculations, however, are not affected by these glitches as the semiconductor equations are solved with the ψ, n and p variable set.

Currently, PISCES-IIB sets the underflow limit at 10^{-38}. If your machine has a larger exponent range, this limit can be change by redefining the ‘mindbl’ and ‘lgmind’ variables in the file pisci.f (Note that ‘lgmind’ is the base 2 logarithm of ‘mindbl’ ; both must be consistent).

0.4.4. Surface Recombination

PISCES-IIA included two basic recombination mechanisms: Shockley-Read-Hall recombination and Auger recombination. Additionally, a special boundary condition for carrier concentrations was set up to include non-infinite surface recombination velocities at some electrodes. It has been found useful to also include an additional recombination component at specific insulator-semiconductor interfaces. This recombination mechanism can be described by a surface recombination velocity as described in [13].

In PISCES-IIB, surface recombination velocities for holes and electrons can be specified for any interface using the INTERFACE card (see appendix). Note that the INTERFACE card also includes fixed charge densities, replacing the old QF card in PISCES-IIA. For each node on the interface so specified, an effective SRH lifetime for each carrier, $\tau^{eff}_{n,p}$, is computed based on the given recombination velocity, $s_{n,p}$, i.e.,

$$\frac{1}{\tau^{eff}_{n,p}(i)} = \frac{A_i}{s_{n,p} d_i} + \frac{1}{\tau^0_{n,p}(i)} \tag{2.35}$$

where $\tau^0_{n,p}(i)$ is the regular SRH lifetime at node i (possibly concentration dependent) and A_i is the semiconductor area and d_i is the length of the interface associated with node i. Different interfaces can be defined separately with arbitrary recombination velocities at each.

0.4.5. Barrier Lowering

As mentioned above, PISCES-IIA included a Schottky boundary condition defined by a work function of the electrode metal and an optional surface recombination velocity. Referring to equations (2.42)-(2.44) in [1],

$$\psi_S = \psi_{S0} = \chi + \frac{E_g}{2q} + \frac{kT}{2q} \ln \frac{N_C}{N_V} - \phi_m + V_{applied} \tag{2.36}$$

$$J_{sn} = q v_{sn} \left(n_S - n_{eq} \right) \tag{2.37}$$

$$J_{sp} = q v_{sp} \left(p_S - p_{eq} \right) \tag{2.38}$$

where $\psi_S = \psi_{S0}$ is the potential at the contact, χ is the electron affinity of the semiconductor, ϕ_m is the work function of the metal, J_{sn} and J_{sp} are the electron and hole contact currents, n_S and p_S are the actual hole and electron concentrations at the contact, n_{eq} and p_{eq} are the equilibrium electron and hole concentrations (infinite recombination velocity) and v_{sn} and v_{sp} are the electron and hole surface recombination velocities, respectively.

In PISCES-IIB, the Schottky model has been enhanced to account for field-dependent barrier-lowering mechanisms, arising from image-forces and possible static dipole layers at the metal-semiconductor interface [14]. Defining the barrier height as

$$\phi_{bn} = \phi_m - \chi \tag{2.39}$$

$$\phi_{bp} = \chi + \frac{E_g}{2q} - \phi_m \tag{2.40}$$

then the amount by which these barriers are lowered is expressed as

$$\Delta\phi_b = \left[\frac{q}{4\pi\varepsilon_s} \right]^{1/2} E^{1/2} + \alpha E \tag{2.41}$$

where E is the magnitude of the electric field at the interface. Note that the term with the square root dependence on electric field corresponds to the image force, while the linear term corresponds to the dipole effect. The coefficient α is user accessible (see the description of the CONTACT card); see [14] for typical values of α.

Barrier lowering had been previously implemented in an experimental copy of PISCES [15]. It has been re-incorporated in PISCES-IIB in a slightly different manner. The basic procedure is to solve Poisson equation normally with the boundary condition $\psi_S = \psi_{S0}$ as in (2.36). However, using the electric field consistent with the solved potentials, effective surface potentials are computed as

$$\psi_{S_{eff}} = \psi_{S0} \pm \Delta\phi_b \tag{2.42}$$

where the + is for electrons and the – for holes, respectively. The continuity equations are then solved using (2.37) and (2.38) as boundary conditions, but with n_S and n_{eq} replaced with $n_{S_{eff}}$ and $n_{eq_{eff}}$ which are computed using $\psi_{S_{eff}}$. The physical interpretation of this approach is that the Poisson equation is solved consistently with the charge, but that the electrons and holes see a combined Poisson and image-force potential. Note that the full barrier lowering term has been applied directly at the surface; in reality this peak occurs a slight distance within the semiconductor.

In PISCES-IIB, surface recombination is implemented on a triangle by triangle basis. That is, using the surface recombination velocity and geometrical data, a recombination component is calculated for each triangle that an element of interest in connected to. Using the electric field for each triangle, an adjusted recombination term can be computed if barrier lowering is to be incorporated. This is in contrast to [15] where a single field value for the electrode node was used to compute a total recombination value. Also unlike [15], barrier lowering can be used with any of the basic numerical solution procedures, i.e. Gummel or Newton.

0.5. Other Modifications

A number of other enhancements of a slightly more minor nature have been made. Apart from those which are relatively transparent (such as modifications to speed up certain parts of the code, etc.), most are briefly mentioned below. See the appropriate card description in the Appendix for a more details.

0.5.1. Ascii file i/o

In almost all cases where PISCES-IIA used to use a binary input or output file (MESH, SOLVE, LOAD, REGRID cards), those files can now be specified as ascii by the inclusion of a logical parameter. This feature is useful in that it then allows transfer of data files between different computer types.

0.5.2. Mesh and doping

Several new features have been added to the MESH, REGRID and DOPING cards. First on the MESH card, the user now has the option of directing the diagonals in a rectangular mesh. The choices are to have all the diagonals face the same way or to have symmetry about the center of the grid as did PISCES-IIA (see Figure 5.9 in [1]). On the REGRID card, the electric field can now be selected as a refinement criterion. On the DOPING card, the latest version of SUPREM-III, supporting the ‘‘export’’ data file format, can be used as input to PISCES. The original version can still be used as well. Finally, PISCES-IIB also allows profiles to be implemented via ascii (concentration versus depth) data files. This feature is useful, for example, if one has experimental profile information.

0.5.3. Plots

A substantial number of enhancements were performed on the PLOT.1D card. A number of new quantities can now be plotted (valence and conduction band potentials, applied and actual biases, x and y components of any vectors as well as a number of AC quantities). A few new control parameters have been added (‘x.log’, ‘unchanged’, ‘x.max’, ‘no.order’ and ‘negative’), and an option to integrate the plotted function is now available (for example, if one integrates a plot of current versus time, the result is charge versus time). Finally, a data point can be skipped in a log file by replacing the blank in the column 1 of the first associated line with a ‘#’.

The PLOT.2D/CONTOUR cards have some new features as well. First, color contour fill has been implemented for the Tektronix 4107 terminal. Instead of simple contour lines, areas of the device are color coded and filled, depending on their values. Second, as with the PLOT.1D card, one can now plot x and y components of vectors separately and can plot valence and conduction band potentials. A ‘negative’ option has also been installed. Third, contours can now be specified by giving a number of contour lines as opposed to an increment. Additionally, the minimum and maximum contour values now default to the minimum and maximum values of the quantity to be plotted. Finally, current flowlines can now be plotted using the algorithm in [16]. An example is shown in Figure 5. Here there are 11 flowlines (specified by setting the number of contours to 11) with 2 invisible because they lie exactly on the outer boundary of the device. Basically the interpretation of Figure 5 is that 10% of the current flows between each line.

0.5.4. Still others

Very briefly, a few more miscellaneous enhancements. The CHECK feature has been substantially enhanced in its output. The PRINT feature (as with PLOT.1D and CONTOUR) can print x and y components of vectors individually. The VECTOR card has an improved syntax and scaling system. The METHOD card allows for the specification of a maximum number of loops for ICCG (this was a fatal error in PISCES-IIA). Finally, the SOLVE card now allows transient ramps as input in a very painless way.

0.6. References

[1] M. R. Pinto, C. S. Rafferty and R. W. Dutton, “PISCES-II - Poisson and Continuity Equation Solver,” Stanford Electronics Laboratory Technical Report, Stanford University, September 1984.

[2] C. S. Rafferty, M. R. Pinto and R. W. Dutton, “Iterative methods in semiconductor device simulation,” *IEEE Transactions on Electron Devices*, October, 1985.

[3] C. H. Price, “Two Dimensional Numerical Simulation of Semiconductor Devices,” Ph.D. Dissertation, Stanford University, May, 1982.

[4] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[5] A. De Mari, “An accurate numerical one-dimensional solution of the *p-n* junction under arbitrary transient conditions,” *Solid-State Electronics*, vol. 11, 1968, pp. 1021-1053.

[6] J. Lambert, *Computational Methods in Ordinary Differential Equations*, Wiley, 1973.

[7] R. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose and R. K. Smith, “Transient simulation of silicon devices and circuits,” *IEEE Transactions on Electron Devices*, October, 1985.

[8] M. R. Pinto and R. W. Dutton, “Accurate trigger condition analysis for CMOS latch-up,” *IEEE Electron Device Letters*, vol. EDL-6, February, 1985.

[9] H. Iwai, M. R. Pinto, C. S. Rafferty, J. E. Oristian and R. W. Dutton, ‘‘Velocity saturation effect on short channel MOS transistor capacitance,’’ *IEEE Electron Device Letters*, vol. EDL-6, March, 1985.

[10] S. E. Laux, ‘‘Techniques for small-signal analysis of semiconductor devices,’’ *IEEE Transactions on Electron Devices*, October, 1985.

[11] Zhiping Yu and R. W. Dutton, *SEDAN III - A Generalized Electronic Material Device Analysis Program*, Stanford Electronics Laboratory Technical Report, Stanford University, July 1985.

[12] W. B. Joyce and R. W. Dixon, ‘‘Analytic Approximation for the Fermi Energy of an ideal Fermi Gas,’’ *Appl. Phys. Lett.*, vol. 31, pp. 354-356, 1977.

[13] A. S. Grove, *Physics and Technology of Semiconductor Devices*, Wiley, 1967.

[14] S. M. Sze, *Physics of Semiconductor Devices*, Wiley, 1981.

[15] E. Sangiorgi, C. S. Rafferty, M. R. Pinto and R. W. Dutton, ‘‘Non-planar Schottky device analysis and applications,’’ Proc. International Conf. on Simulation of Semiconductor Devices and Processes, Swansea, U.K., July, 1984.

[16] E. Palm and F. Van De Wiele, ‘‘Current lines and accurate contact current evaluation in 2D numerical simulation of semiconductor devices,’’ *IEEE Transactions on Electron Devices*, October 1985.

Chapter 3

General Discussion

0.1. Numerical Considerations

In this section, some additional numerical considerations will be discussed. Refer to [1,2] for background material and additional details.

0.1.1. Computational methods

PISCES-IIA served as both a general two-dimensional device simulation code and as a vehicle to explore various alternative computational methods [2]. PISCES-IIA had four basic numerical solution techniques as described in [1]: the Gummel method, block-Newton method, knot-iterative method and full Newton method. From the experiences gathered since the initial release of the program on numerous computers and applications, the full Newton and Gummel methods should now be used exclusively. Hence, only these two methods have been included in PISCES-IIB. Also, only the (ψ, n, p) variable set has been implemented in PISCES-IIB.

As will be shown below, the Gummel and full-Newton methods have fairly distinct regions of preference. In general, the Gummel method is preferred for zero/reverse bias and low-level injection simulations. The full Newton is preferable for high-level injection as well as MOS saturation conditions (see [1]); it is required for transient and ac small-signal analysis. As an brief illustration, Figure 1a shows a cross-section of a submicron bipolar transistor [3], suitable for comparing the computational methods in PISCES-IIA. Because of the aggressive scaling of lateral dimensions, this device shows strong two-dimensional effects, particularly with regard to collector resistance and current crowding. Figure 1b shows a close-up of the intrinsic region of the device, with a base-emitter bias of 0.9V. Contours of hole current density are plotted at intervals of $2x10^3$Amps/cm^2. The base current is dominated by injection through the corners of the emitter, degrading the gain. Such effects can be reduced by accurate control of the emitter and extrinsic base profiles.

This bipolar transistor was simulated on a grid of approximately 1300 points, stepping the base-emitter voltage in 100mV increments initially and 25mV increments later upon achieving very high forward bias. Identical simulations were performed using Gummel's method, the block method, the knot method and the full Newton (sparse direct LU) including Newton-Richardson (see below). Figures 2a and 2b illustrate the performance of each method. In low-level injection, the block-Newton method performs very well compared to the full Newton; however, as is typically the case, Gummel's method is still better in this bias range. As the built-in voltage of the base-emitter junction is approached ($\approx$ 0.9V), Both the Gummel and block methods become rapidly more expensive, and the block method eventually is a bit faster than Gummel. In contrast, the full Newton is much more insensitive to bias condition. Although slightly more expensive for low-level injection, the full Newton definitely becomes the method of choice for biases above $\approx$ 0.8V where it becomes 4-5 times cheaper than the Gummel or block methods. Finally, the knot-iterative method is generally stable, but consistently more expensive than the direct method. However, it is still felt that this method may hold some promise for very large (3D?) problems with a code optimized for a vector/parallel machine.

0.1.2. Norms

Another important numerical consideration is the convergence criterion used to define the end of the non-linear iteration procedure. PISCES-II has a number of different convergence criteria, as well as error measures for the non-linear iterations. The default error measure is called the ‘‘xnorm’’ which measures the size of the updates (absolute change in potential scaled by kT/q and relative changes in carrier concentrations) on each iteration. The default convergence criterion is that the xnorm for each variable must be less than 10^{-5}. This requirement is VERY conservative, and by no means should it be taken as an absolute! Another warning regarding this error norm: it is not always guaranteed to decrease from iteration to iteration, although the system may be converging.

Possibly a more interesting norm (and one that is guaranteed to go down from one iteration to the next) is the norm of the right-hand side of the non-linear Newton system (‘‘rhsnorm’’). The rhsnorm is basically a measure of the maximum amount by which the net flux or current is out of balance at any single grid point in the mesh. In PISCES, the rhsnorm has been scaled so that it has the same units (coul/μm or Amps/μm) as the terminal fluxes and currents. It is particularly useful to examine the continuity rhsnorms for very low current bias conditions, as they will give some information on the resolution obtained; if the rhsnorms are on the same order as any terminal currents of interest, those currents may not be reliable. For such cases, it may be advantageous to force one or two more iterations to see if the norm can be driven down further. Although the rhsnorm is not generally used as a strict criterion for convergence (the xnorm tends to be more reliable in this regard), it is generally advisable to print it particularly if convergence problems are suspected. Note that if both ‘‘xnorm’’ and ‘‘rhsnorm’’ are set on the METHOD card, both will be printed, but the xnorm will be used exclusively to determine convergence.

0.1.3. Initial guesses

Another area of importance with respect to the convergence of the non-linear iteration is the quality of the initial guess used to start the procedure. A good initial guess may cut down the number of iterations by factors of two or more. As described in [1], PISCES-II has basically four different kinds of initial guesses† : ‘initial’, ‘previous’, ‘project’ and ‘local’, any one of which can be selected as an option on the SOLVE card. *Initial* is used to start a new device; this guess is obtained by assuming charge neutrality throughout the device. Generally it is used to obtain a solution with zero applied bias. *Previous* takes the solution at the previously solved bias point and adjusts the boundary conditions (if necessary) only. Every internal ψ, n, p is left as it was for the previous point. *Project* extrapolates internal ψ, n, p from two previous bias points. It therefore requires two previous solutions, which additionally must have equivalent bias steps taken on any electrodes that have had there voltages changed (the bias step may be different from one solution to the next however). Finally, the *local* guess uses the majority-carrier quasi-Fermi potential (as determined by the applied biases), to set ψ, n, p.

Given that *initial* will only be used for the first bias point, there remains a choice of *previous*, *project* or *local* for any other bias points. *Local* tends to work best for strictly reverse-biased devices and is especially efficient when trying to jump by very large voltage increments. *Project* is usually better than *previous* where applicable and therefore is the default in situations where either *project* or *previous* can be used. *Previous* is the conservative choice. It is required for time-dependent simulation and current sources.

There are situations that arise where the user can force PISCES to do better than it would have automatically. A case in point is a bipolar transistor where one is stepping V_{BE} with a constant V_{CE}. As shown in Table I, *previous* outperforms *project* in high-level injection. The performance of a current boundary condition on the base is also shown.

† Note that as described in [1], there is one other type of initial guess that is generated after a re-grid of a mesh upon which a solution had existed in order to estimate values for newly created nodes.

Table I - Newton loops for a bipolar transistor

V_{BE}	I_B	I_C	*Previous*	*Project*	Current
0.1v	$-3.11\text{x}10^{-14}$	$8.50\text{x}10^{-14}$	7	-	7
0.2v	$-2.58\text{x}10^{-14}$	$1.04\text{x}10^{-13}$	7	4	9
0.3v	$1.85\text{x}10^{-14}$	$3.36\text{x}10^{-12}$	7	4	12
0.4v	$6.71\text{x}10^{-13}$	$1.53\text{x}10^{-10}$	7	5	>20
0.5v	$2.11\text{x}10^{-11}$	$6.98\text{x}10^{-9}$	7	5	>20
0.6v	$8.96\text{x}10^{-10}$	$3.10\text{x}10^{-7}$	8	4	>20
0.7v	$3.67\text{x}10^{-8}$	$9.51\text{x}10^{-6}$	8	5	>20
0.75v	$2.09\text{x}10^{-7}$	$3.35\text{x}10^{-5}$	7	5	7
0.8v	$9.93\text{x}10^{-7}$	$8.18\text{x}10^{-5}$	7	6	8
0.825v	$2.10\text{x}10^{-6}$	$1.15\text{x}10^{-4}$	6	7	7
0.85v	$5.12\text{x}10^{-6}$	$1.60\text{x}10^{-4}$	8	9	8
0.875v	$1.34\text{x}10^{-5}$	$2.20\text{x}10^{-4}$	9	>20	9
0.9v	$2.88\text{x}10^{-5}$	$2.92\text{x}10^{-4}$	9	>20	9

The situation actually improves for each case as the device moves further into high-level injection; for instance, in going from $V_{BE} = 1.2V$ to $V_{BE} = 1.3V$, *previous* only takes 6 Newton loops while *project* takes 10. Remember that *project* is the default automatically selected by PISCES for each bias point after $V_{BE} = 0.1V$. Another common situation where a previous guess is better than project is in stepping the drain of a MOSFET with gate held constant. For that reason, it is often more economical to compute MOSFET characteristics by stepping the gate with constant drain, using project.

Another alternative to improving initial guesses is of course to simply take smaller bias steps. Adjacent solutions then become much more similar, and the number of Newton loops *per bias point* is reduced when using either *project* or *previous*. However, since more bias points need to be solved for, the total computation time may increase. A procedure exists within PISCES to reduce the size of the bias step taken if non-convergence is observed (see the parameters ‘‘trap’’ and ‘‘atrap’’ on the METHOD card). This feature has been found of particular value when finding knees of I-V curves with voltage boundary conditions.

0.2. Increasing Bounds

As mentioned above, the Gummel and full Newton methods have been found preferable for nearly all bias conditions and are therefore to be used exclusively in PISCES-IIB. The Gummel method is not very memory intensive; currently it can handle up to 3000 grid points. The full Newton method is VERY memory intensive. PISCES-IIB as configured prohibits the use of more than 1600 points with the full Newton and 2 carriers (2000 points for the full Newton and one carrier). These bounds are somewhat arbitrary. They were chosen so that the entire program would be about 7-MByte in order to fit in non-virtual 8-Mbyte machines.

Given that most users do have access to machines/operating systems with virtual memory, there has been a desire expressed to increase the maximum number of grid points that can be treated. In this section, a few comments and guidelines will be presented that may help in this regard. One word of caution: applications will tend to expand to fill the number of grid points allowed, regardless of whether the points are necessary or not (a corollary to one of Murphy’s laws, so we are told). Especially now that options such as lumped elements are available, every possible objection should be raised to the use of multi-thousand node simulation runs if you share a computer resource amongst several PISCES users. It may be advisable to actually support different sized versions of the code to cut down on the amount of memory requested at one time.

The majority of the memory required by PISCES is involved with the sparse LU solver. PISCES uses the sparse solver VEGES [4] along with a minimum degree ordering from YSMP[5]. An explanation of the functions of the various arrays involved in these codes is beyond the scope of this report; see [4,5,6] for details. However, it should be possible to increase the bounds of these arrays without a detailed knowledge of their purpose.

PISCES has a common block called ‘emaco’ which contains most of the large arrays used during execution. This common block has a static (permanent) part in the include file ’com.emaco’. During various phases of the program, other temporary storage is added to the end of this static part by an include of another common file following ‘com.emaco’. These additional files usually have the letters ‘tmp’ in their name; for instance, during a solution using the full Newton method, the temporary storage is in ‘com.soltmpn’ while during a plot, ‘com.plttmp’ is used.

The declared dimensions within PISCES are stored in a static common block called ‘symco’ in the include file com.symb. For instance, ‘mp2fn’ is the maximum number of points for Newton with 2 carriers, ‘mp1fn’ is for Newton with 1 carrier and ‘mpgum’ is for Gummel. These variables and others (which are generally self-explanatory) are set in the block data subprogram.

All arrays within PISCES that are dimensioned to 3000 are dependent on the maximum number of points. There are also a number of arrays that have a dimen sion of 6000, referring to the maximum number of triangles, which is always less than two times the number of grid points. There are two other similar dependencies which are not as obvious. In ‘com.rgdtmp’, the arrays dimensioned to 12000 refer to twice the maximum number of triangles, while in ’com.soltmpn’, the arrays dimensioned to 9800 refer to three times the maximum number of points allowed with the full Newton and 2 carriers (mp2fn which defaults to 1600).

The sparse matrix arrays are quite a bit harder to set dimensions for. In addition to numeric arrays for the matrix to be factored, A, and its upper and lower triangular factorization, L and U, PISCES requires a set of pointer arrays for each of these (IA, JA, IVA, IL, JL, IVL, IU, JU, IVU). These pointer arrays are produced, along with the pivoting order (IPC and IPRI), during the symbolic factorization and are kept in ‘com.emaco’. The numeric A, L and U are only used during the solution phase of PISCES and are kept in temporary storage, as are all the various arrays associated with performing the symbolic factorization.

The pointer arrays JA, JL, JU and IVA, IVL, IVU are the relatively straight forward to dimension; they are set to the maximum number of equations to be solved simultaneously which is max(3*mp2fn,2*mp1fn,mpgum)+1. Therefore, JA, etc. are now set to 3*1600+1=4801.

The size of IA, IL, IU, A, L and U are somewhat harder to determine. In fact, they are quite structure dependent. The best advice for setting these quantities is to run a set of examples with varying number of points using the ‘print’ option on the SYMBOLIC card which will report on the actual size of each array. Table II shows the result of such an experiment for a square device (rectangular grid). Note that NP is the number of points and NEQ is the number of PDEs solved simultaneously; i.e., NEQ=1 refers to 0 carrier or Gummel method simulations and NEQ=2 or 3 refers to 1 or 2 carrier Newton simulations. Similarly, Table III shows the results of an identical experiment using a ‘‘thin’’ device, i.e. one that is rectangular but only 5 nodes high. There are several interesting features to note in Tables II and III. First, there is about a factor of 2 difference in storage for A in going from 1 to 2 equations (0 carriers to 1 carrier, full Newton) and a factor of 3 difference in going from 1 to 3 equations (0 carriers to 2 carriers, full Newton). The size of A seems to be proportional to the number of equations being solved for simultaneously. Second, in Table II the sizes of L and U, which are identical, seem to be proportional to the square of the number of equations; hence increases of 4 and 9 times are observed for 1 and 2 carrier full Newton simulations respectively over 0 carrier simulations. Third, note that because of fill-in and the vector pointer scheme used in the sparse solver, the sizes of L and U become much larger than their associated pointers IL and IU. This size differential becomes even more significant if data types are considered since L and U must be real*8, while IL and IU can be integer*2. Finally, the

Table II - Array sizes for square device (n-by-n)

NP=n^2	NEQ	IA	IL	IU	A	L	U
25	1	91	63	57	105	78	78
100	1	381	389	394	460	558	558
400	1	1561	1989	2087	1920	3392	3392
900	1	3541	5024	5362	4380	9712	9712
1600	1	6321	10006	9878	7840	19766	19766
25	2	261	191	165	420	334	334
100	2	1121	1214	1256	1840	2325	2325
400	2	4641	5955	6283	7680	13965	13965
900	2	10561	15760	15646	17520	39685	39685
1600	2	18881	30097	28873	31360	80797	80797
25	3	391	328	265	945	769	769
100	3	1691	2002	1978	4140	5305	5305
400	3	6961	9748	9847	17280	31720	31720
900	3	15841	26506	24448	39420	89965	89965
1600	3	28321	51223	45232	70560	183046	183046

Table III - Array sizes for thin device (n-by-5)

NP=5n	NEQ	IA	IL	IU	A	L	U
25	1	91	63	57	105	78	78
100	1	391	320	282	450	412	412
400	1	1591	1325	1152	1830	1732	1732
900	1	3591	3000	2602	4130	3932	3932
1600	1	6391	5345	4632	7350	7012	7012
25	2	261	191	165	420	334	334
100	2	1161	971	871	1800	1745	1745
400	2	4761	4155	3631	7320	7325	7325
900	2	10761	9451	8231	16520	16625	16625
1600	2	19161	16875	14671	29400	29645	29645
25	3	391	328	265	945	769	769
100	3	1741	1660	1387	4050	4000	4000
400	3	7141	7147	5797	16470	16780	16780
900	3	16141	16300	13147	37170	38080	38080
1600	3	28741	29107	23437	66150	67900	67900

distinct difference between the thin and square devices is noted in the sizes of L and U. Although the A dependence is roughly identical, the sizes of L and U are much smaller for the thin device;

in fact, the sizes of L and U are almost equal to A. This result is not surprising due to the extremely low amount of fill-in that can be achieved for such structures using a minimum degree pivot ordering. The unfortunate consequence of such a result is that different device geometries will have different memory requirements. Such considerations must be addressed in choosing array bounds. Note further that pure triangular grids (see chapter 4), as opposed to the rectangular grids above, will also yield slightly different bounds for a given number of nodes.

There are a number of other (temporary) arrays during symbolic that are located in com.symtmp1, com.symtmp3 and com.symtmp4. The actual sizes of most of these are also printed out using the ‘‘print’’ option, although they were not shown above. Additionally, there may be some arrays with mysterious bounds in some common blocks. Some of the more obvious bounds are : the maximum number of regions (8), the maximum number of electrodes (10), the maximum plot buffer length (1000), and the maximum number of electrode nodes (300). Any other dimensions probably need never be changed, and hopefully, if any are ever exceeded, PISCES should respond gracefully with a suitable diagnostic message.

One final comment should be made on the sparse matrix package. The LU factorization (subroutine VMNPD) is the most costly (in cpu) section of the program. As such, it might be worth trying to optimize if one has access to a vector machine. However, this optimization process is far from being trivial. In order to gain substantial benefit, one must really understand the sparse matrix pointer structure and operations. Refer to [7] for a possible improvement and a discussion of some of the issues involved.

0.3. Model Implementation

Another area where some user customization has been desired is in the implementation of new physical models. Hence, this section includes a brief discussion of the general issues and approach to such an endeavor.

Referring to Figure 3, we write the box discretization [8] for a node in the triangular grid:

$$\sum_t \left(-\varepsilon \frac{h_3}{d_3}(\psi_2 - \psi_1) - \varepsilon \frac{h_2}{d_2}(\psi_3 - \psi_1) - \rho_1 A_1 \right) = \sum_t F_{\psi_1}(t) = 0 \tag{3.1}$$

$$\sum_t \left(J_{n_{12}} h_3 + J_{n_{13}} h_2 - qU_1 A_1 \right) = \sum_t F_{n_1}(t) = 0 \tag{3.2}$$

$$\sum_t \left(J_{p_{12}} h_3 + J_{p_{13}} h_2 + qU_1 A_1 \right) = \sum_t F_{p_1}(t) = 0 \tag{3.3}$$

The sum in (3.1)-(3.3) is over all the triangles that the node labeled ‘1’ is a part of. The indices ‘1’,‘2’ and ‘3’ in (3.1)-(3.3) should more properly have a superscript t, with ‘1^t’ always being the node of interest and ‘2^t’ and ‘3^t’ the nodes adjacent to ‘1^t’ for each triangle t (see figure 3b). $J_{n,p_{12}}$ and $J_{n,p_{13}}$ above refer to the current densities between nodes 1 and 2 and between nodes 1 and 3 of a triangle t and are computed using the Scharfetter-Gummel discretization [9]:

$$J_{n_{12}} = \frac{q\mu_{n_{12}}}{d_3} \left[(n_2 - n_1)\mathbf{B}(\psi_2 - \psi_1) - n_1(\psi_2 - \psi_1) \right] = \mu_{n_{12}} G_{n_{12}} \tag{3.4}$$

$$J_{p_{12}} = \frac{q\mu_{p_{12}}}{d_3} \left[(p_1 - p_2)\mathbf{B}(\psi_2 - \psi_1) - p_2(\psi_2 - \psi_1) \right] = \mu_{p_{12}} G_{p_{12}} \tag{3.5}$$

where **B** is the Bernoulli function.

The equation assembly procedure within PISCES is performed on a triangle by triangle basis. That is, a submatrix and right-hand side are set up for each triangle, containing the equations relating the nodes of that single triangle. Afterward, this submatrix and right-hand side are added to the global system. The submatrix (estifm) has the following form:

$$\text{estifm} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \frac{\partial F_1}{\partial x_3} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \frac{\partial F_2}{\partial x_3} \\ \frac{\partial F_3}{\partial x_1} & \frac{\partial F_3}{\partial x_2} & \frac{\partial F_3}{\partial x_3} \end{bmatrix} \tag{3.6}$$

where, for the full Newton and 2 carriers,

$$\frac{\partial F_i}{\partial x_j} = \begin{bmatrix} \frac{\partial F_{\psi_i}}{\partial \psi_j} & \frac{\partial F_{\psi_i}}{\partial n_j} & \frac{\partial F_{\psi_i}}{\partial p_j} \\ \frac{\partial F_{n_i}}{\partial \psi_j} & \frac{\partial F_{n_i}}{\partial n_j} & \frac{\partial F_{n_i}}{\partial p_j} \\ \frac{\partial F_{p_i}}{\partial \psi_j} & \frac{\partial F_{p_i}}{\partial n_j} & \frac{\partial F_{p_i}}{\partial p_j} \end{bmatrix} \quad i = 1, 2, 3 \ \ , \ \ j = 1, 2, 3 \tag{3.7}$$

The corresponding right-hand side for the triangle, wrhs, is

$$\text{wrhs} = -\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \tag{3.8}$$

where, again for the full Newton method and two carriers,

$$F_i = \begin{bmatrix} F_{\psi_i} \\ F_{n_i} \\ F_{p_i} \end{bmatrix} \quad , \quad i = 1, 2, 3 \tag{3.9}$$

Therefore, estifm is a 9-by-9 matrix and wrhs has a length of 9 for the full Newton method and two carriers. Note that for the full Newton and only one carrier, $\partial F_i / \partial x_j$ would only be 2-by-2 and F_i would be of length 2, so estifm would be 6-by-6 and wrhs would be of length 6. For 0 carriers and Gummel, estifm is a simple 3-by-3 matrix, and wrhs is a vector of length 3.

In order to assemble the equations for a single triangle t, all the elements of estifm and wrhs must be computed. By using (3.8) and (3.9) along with (3.1)-(3.5), wrhs can be constructed directly. However, estifm requires careful calculation of derivatives. Failure to include proper derivative terms, either by neglect or error, may lead to slow convergence (or divergence). Regardless of the exactness of estifm, if the iterations do converge, the solution will be correct if wrhs is correct.

As an example, $\partial F_{n_1} / \partial n_2$ will be formulated. This entry corresponds to estifm(2,5) for the full Newton with two carriers, estifm(2,4) for the full Newton with one carrier and estifm(1,2) for the Gummel method. The corresponding right-hand side entry (wrhs(2) for Newton with one or two carriers and wrhs(1) for Gummel) is

$$-F_{n_1} = -\left(h_3 J_{n_{12}} + h_2 J_{n_{13}} + qA_1 U_1\right) \tag{3.10}$$

Therefore,

$$\frac{\partial F_{n_1}}{\partial n_2} = h_3 \frac{\partial J_{n_{12}}}{\partial n_2} + h_2 \frac{\partial J_{n_{13}}}{\partial n_2} + qA_1 \frac{\partial U_1}{\partial n_2} \tag{3.11}$$

Substituting in (3.4),(3.5) and noting that $\partial J_{n_{13}} / \partial n_2 = 0$,

$$\frac{\partial F_{n_1}}{\partial n_2} = h_3 \left[\mu_{n_{12}} \frac{\partial G_{n_{12}}}{\partial n_2} + G_{n_{12}} \frac{\partial \mu_{n_{12}}}{\partial n_2} \right] + qA_1 \frac{\partial U_1}{\partial n_2} \tag{3.12}$$

There are three derivatives in (3.12) to compute. The first, $\partial G_{n_{12}} / \partial n_2$ can be computed directly from (3.4):

$$\frac{\partial G_{n_{12}}}{\partial n_2} = \frac{q}{d_3}\mathbf{B}(\psi_2 - \psi_1) \tag{3.13}$$

Recombination (SRH or Auger) is generally calculated at a node directly from the carrier concentrations at that node, as is the space charge, ρ_1 in (3.1). In this case, $U_1 = f(n_1, p_1)$, therefore

$$\frac{\partial U_1}{\partial n_2} = 0 \tag{3.14}$$

Note however that if impact-ionization, generally is written as a function of the hole and electron currents $J_{n_{12}}$ and $J_{p_{12}}$ [10], was to be added, the generation term produced would have a dependence on n_2 from the diffusion component of $J_{n_{12}}$.

Finally in (3.12), $\partial\mu_{n_{12}}/\partial n_2$ must be evaluated. Within PISCES, mobility presently can have several dependencies (see chapter 2 of [1]). Some dependencies may be static; for instance, mobility may be a function of impurity concentration which does not change with the solution. Additionally, mobility may have a dependence of electric field and hence potential, introducing a non-zero $\partial\mu_{n_{12}}/\partial\psi_1$ and $\partial\mu_{n_{12}}/\partial\psi_2$. Suppose it was desirable to include some carrier-carrier scattering mechanism within PISCES (there currently is no such model within PISCES-IIB). Just for illustration, assume the mobility should then be written as

$$\mu_{n_{12}} = \mu^0_{n_{12}} f(n_1, n_2, p_1, p_2) = \mu^0_{n_{12}}\left[1 + C(n_1 + n_2 + p_1 + p_2)\right]^{-1/2} \tag{3.15}$$

where $\mu^0_{n_{12}}$ may include both static and field dependencies and C is a constant. Then,

$$\partial\mu_{n_{12}}/n_2 = -\frac{C}{2}\,\mu^0_{n_{12}}\left[1 + C(n_1 + n_2 + p_1 + p_2)\right]^{-3/2} \tag{3.16}$$

Therefore, for this mobility model, we obtain

$$\frac{\partial F_{n_1}}{\partial n_2} = h_3\left[\frac{q\mu_{n_{12}}}{d_3}\mathbf{B}(\psi_2 - \psi_1) - \frac{CG_{n_{12}}}{2}\,\mu^0_{n_{12}}\left[1 + C(n_1 + n_2 + p_1 + p_2)\right]^{-3/2}\right] \tag{3.17}$$

where $G_{n_{12}}$ is computed from (3.4) and $\mu_{n_{12}}$ is computed from (3.15). Any other of the elements of estifm (and wrhs) are assembled in a similar manner.

One final point to consider is that throughout the PISCES code, variables and equations are scaled. This scaling takes place in a subroutine called MATER. Basically, any potentials are scaled by kT/q while concentrations (including doping) are scaled by n_i. Poisson's equation, (3.1), is scaled by a variable called ‘‘decoef’’, while the continuity equations, (3.2) and (3.3) are scaled by a variable called ‘‘djcoef’’. See MATER for the precise values for these coefficients and any other scaling information.

0.4. Vector Calculations

It is sometimes useful to examine internal fields or currents from a PISCES solution. Unfortunately these vector quantities, unlike the real solution (ψ, n, p), are not accurately defined at grid points but instead are computed between nodes along the edges of triangles. (see (3.4) and (3.5)). This presents a problem whenever internal currents or fields are to be printed or plotted, in that there is no obvious way to assign a nodal value of current or electric field. Further, in order to interpolate contours, it is required to obtain nodal values for currents and fields.

However, with some mathematical manipulation, reasonable nodal averages can be obtained for these vector quantities. In the following, the electric field will be used as an example, but any of the various current components (or total current) can be treated in a similar fashion.

Consider the expression

$$\nabla\cdot(\lambda\vec{E}) = \lambda\nabla\cdot\vec{E} + \vec{E}\cdot\nabla\lambda \tag{3.18}$$

where $\vec{E} = -\nabla\psi$ is the electric field and λ is some scalar quantity. Noting that from Poisson's equation (2.1)

$$\nabla \cdot \vec{E} = \frac{\rho}{\varepsilon} \tag{3.19}$$

(3.18) can be simplified to

$$\nabla \cdot (\lambda \vec{E}) = \lambda \frac{\rho}{\varepsilon} + E_x \frac{\partial \lambda}{\partial x} + E_y \frac{\partial \lambda}{\partial y} \tag{3.20}$$

where E_x and E_y are the x and y components of the electric field. Now if one arbitrarily chooses $\lambda = x$ and re-arranges (3.20), then

$$E_x = \nabla \cdot (x\vec{E}) - x \frac{\rho}{\varepsilon} \tag{3.21}$$

Using Gauss's law just as with the Poisson equation itself [8], (3.21) can be used to obtain a reasonable estimate of E_x at each node. Similarly, with $\lambda = y$, E_y and hence $\vec{E}$ can be computed. Note that identical arguments hold true for currents as well, with the added benefit that when calculating total node current, $\nabla \cdot \vec{J}_{TOTAL} = 0$ so that the first term on the right-hand side of (3.18) is zero.

It must be stressed that the final results are still just estimates; if (3.21) is fully integrated and expanded, one finds that the result is a weighted average of x and y components by triangle area. For more exact applications, a self-consistent electric field value is available for each triangle (constant over the triangle) computed from the three potentials at each triangle vertex. This is in fact how barrier-lowering computations are performed - see section 4.3 of chapter 2.

0.5. Errata

Since this report is to be a supplement and not a replacement for our earlier document, the errors found to date in [1] are listed below:

- The ‘‘ascii’’ parameter on the PLOT.1D card is a logical flag and not a character string; i.e., if an output file is requested using the ‘‘outfile=<ch string>’’ parameter, it is binary by default and ascii only if the ‘‘ascii=<logical>’ is set to true.
- The example in OPTIONS card section is incorrect. It should read:

  ```
  OPTIONS   TEK4107  X.S=6 ...
  ```
  not
  ```
  OPTIONS   DEV=TEK4107  X.S=6 ...
  ```
- An example in EXTRACT card section is incorrect. It should read:

  ```
  EXTRACT   METAL.CH CONT=1 ...
  ```
  not
  ```
  EXTRACT   ME.CHARGE CONT=1 ...
  ```
- No fixed charge density was supplied for the QF card example.
- The defaults for ‘‘y.top’’ and ‘‘y.bot’’ are listed incorrectly in the section on the DOPING card for y-direction profiles. The correct defaults are ‘‘SP’’ for the ‘‘y.bot’’ parameter and ‘‘y.bot’’ for the ‘‘y.top’’ parameter.
- The ‘‘diff’’ option on the LOAD card produces the difference between the solution in file 1 and the solution in file 2, i.e. (solution 1) - (solution 2). The example regarding the ‘‘diff’’ option should therefore have read ‘‘OUTD=SOL1-2’’ not ‘‘OUTD=SOL2-1’’.
- Equation (2.36) is incorrect (the exponentiation operator was left out of the denominator). It should read

$$v_{sat}(T) = \frac{2.4 \times 10^7}{1 + 0.8 \exp\left(\frac{T}{600}\right)}$$

The following features were undocumented:

- When using the Newton-Richardson option for the full Newton method (‘‘autonr’’ on the METHOD card), PISCES will put an asterisk (‘‘*’’) in front of any Newton iteration where the Jacobian was re-factored.
- The analytic concentration-dependent, low-field mobility model for Si (including temperature effects - see section 3 of chapter 2) was not included in the documentation of the MODELS card in [1]. It is included in this report.
- The ‘‘print’’ option for printing model coefficients, flags and physical constants was not included in the documentation of the MODELS card in [1]. It is included in this report.
- No mention was made of the ‘‘tt’’ files (‘‘tt’’ is an abbreviation for triangle-tree) created during each regrid. These files contain a history of the procedure by which an existing mesh was created. Retaining the mesh history helps prevent the occurence of unduly large or small angles during the regrid procedure. It is possible, but inadvisable to regrid a mesh without a history file. A history file is generated for every file that is a regrid of some other mesh; hence, an initial mesh will not have a ‘‘tt’’ file.
- When printing information on grid points (the ‘‘points’’ option on the PRINT card), PISCES sometimes prints some confusing numbers under the electrode column. If the number outputted is positive, it corresponds to the electrode number to which that node is a part. If a negative number is printed, the absolute value of this number (if $\leq$ the total number of electrodes, nelect) corresponds to the electrode to which this node is connected via semiconductor of equivalent doping type. If a negative number is printed whose absolute value is greater than nelect, then this node is connected via equivalent doping to more than one electrode. The absolute value then corresponds to a code in base nelect+1 to indicate each contact association. This association of internal nodes to contacts is necessary for setting quasi-fermi levels properly for 0 and 1 carrier solutions. Additionally, if the number in the electrode column is zero, then this node is part of a doping region which is not contacted by an electrode (or is an insulator node).
- Device-independent plot files can be saved instead of writing to the user’s screen by specifying a binary ‘‘outfile’’ on the PLOT.2D or PLOT.1D cards. These files are useful for off-line or hardcopy plots. Defined here is the file format for programmers who need to write off-line plot drivers.

  The plot files contain a sequence of fixed-length binary records, each containing two real numbers (the x,y coordinates) and an integer (the action code). The actions are:

Code	Action to be performed
-i	Change linetype to i (0<i<8) (x,y are ignored)
0	Initialize the plotting system (x,y ignored)
1	Clear screen / new page (x,y ignored)
2	Draw line to x,y
3	Move to x,y
13	Offset subsequent positions by x,y
15	Rotate subsequent positions around x,y
17	Rotate subsequent positions by x (y ignored)
19	Post buffers (x,y ignored)
20	Obsolete - ignore
40,41	Switch to/from graphics mode (terminals only)
99	Finish plotting

Most of the actions are graphics primitives. The exceptions are the translation and rotation actions which affect the interpretation of all subsequent x,y coordinates. These transformation actions are intended to be cumulative in effect. Note that a rotation is defined by two adjacent calls, one to define the location, the second to define the angle.

0.6. References

[1] M. R. Pinto, C. S. Rafferty and R. W. Dutton, ‘‘PISCES-II - Poisson and Continuity Equation Solver,’’ Stanford Electronics Laboratory Technical Report, Stanford University, September 1984.

[2] C. S. Rafferty, M. R. Pinto and R. W. Dutton, ‘‘Iterative methods in semiconductor device simulation,’’ *IEEE Transactions on Electron Devices*, October, 1985.

[3] E. Crabbé, Stanford University, private communication.

[4] D. A. Calahan and P. G. Buning, ‘‘Vectorized General Sparsity Algorithms with Backing Store,’’ SEL Report 96, Systems Engineering Laboratory, University of Michigan, Ann Arbor, MI, 1977.

[5] S. C. Eisenstat, M. C. Gursky, M. H. Scultz, A. H. Sherman, Dept. Computer Science, Yale University, Technical Report 112, 1977.

[6] A. George and J. W. H. Liu, *Computer Solutions of Large Sparse Positive Definite Systems*, Prentice-Hall, 1981.

[7] R. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose and R. K. Smith, ‘‘Transient simulation of silicon devices and circuits,’’ *IEEE Transactions on Electron Devices*, October, 1985.

[8] C. H. Price, ‘‘Two-dimensional Numerical Simulation of Semiconductor Devices,’’ Ph.D. dissertation, Stanford University, May, 1982.

[9] D. Scharfetter and H. K. Gummel, ‘‘Large-signal analysis of a silicon Read diode oscillator,’’ *IEEE Transactions on Electron Device*, vol. ED-16, pp. 64-77, 1969.

[10]
C. R. Crowell and S. M. Sze, ‘‘Temperature dependence of avalanche multiplication in semiconductors,’’ *Applied Physics Letters*, vol. 9, no. 6, pp. 242-244, 15 September, 1966.

Chapter 4

Grid Generator

0.1. Introduction

This release of PISCES-II introduces a new grid generator. It is written in the C language, and is much more portable than the Pascal program provided in the first release (IGGI). The triangulation algorithm, due to R.E. Bank[1], is new and automatically smooths the grading from coarse to fine areas of the grid. This relieves the user of much of the work in generating a grid. The grid generation process has been split into two phases, editing and triangulation. There are separate programs, named **skel** and **tri** for each phase. **Skel** is an interactive editor where the user defines the skeleton of a structure. **Tri** is the triangulation phase, where the skeleton is filled with triangles. The programs are treated separately in the documentation that follows. The chapter is divided into three sections: a lengthy description of **skel**, then a Unix-style manual page for each program.

0.2. Grid Editing with Skel

0.2.1. Getting started

This section is an introduction to **skel**. The discussion will make more sense if you are running **skel** on a graphics terminal while you are reading this. Sections 1 through 6 discuss the main features of the editor, and section 7 discusses the connection between **skel**, **tri**, and PISCES-II. Finally Section 8 offers some suggestions on using **skel**.

0.2.1.1. Specifying terminal type

Before you start **skel** you must tell it what terminal you are using. This is done (in Unix) by setting the environment variable TERM. At present only the following combinations are supported:

Output device	Input device	Code
HP2623	hp2623 cursor	TERM=hp2623
HP2648	hp2648 cursor (coming soon)	TERM=hp2648
9000 console	HP9111 tablet	TERM=hp9020
TEK4107	tek mouse	TERM=tek4107
TEK4107	tek tablet	TERM=tekt

Your terminal type is normally defined when you log in.

It is important that you define the right terminal before starting **skel**, because the graphics initialization sequence for one terminal will generally cause another to lock up, display strobe lights, or self-destruct. In particular, you need to be specific about which of a manufacturer's models you are using, because the screens are different sizes for (say) a HP2648 and a HP2623.

0.2.1.2. The Display

Assuming the initialization went correctly, both the alpha and graphics terminal screens should clear, a large box with axes should appear on the right of the screen, and several small boxes with text should appear on the left. The right box is the *mesh* window. Initially it is empty, and has a range of roughly 0-1 in both x and y directions. The exact numbers depend on the dimensions of the screen; the bounds are adjusted so that an inch of screen represents the same length regardless of its direction.

The left boxes constitute the command menu, and the text labels are the commands that are available. All the commands have operands; when a command is chosen, a second menu is displayed to the right of the first, and its boxes display the possible operands for that command. Above the command menu is an asterisk, indicating that **skel** is waiting for input from that menu. When input is expected from the operand menu, the asterisk will move to that menu, and when input is expected in the mesh window, the asterisk will be replaced by a longer text over the mesh. By observing which of the windows is labeled, you can always tell what state the program is in. The moving label is referred to as the run light.

One other area of the display is used, and is initially blank. This is the top line of the display, used for text input/output and printing warning messages. When several messages arrive in this buffer in succession, the program will make sure you have read each message before displaying the next by prompting with the message ‘‘--More--’’. The next message can be displayed by typing a carriage return.

0.2.1.3. Input

Most interaction with **skel** is by moving the graphics cursor to some location on the screen and indicating that you have finished moving it. Using cursor keys, the cursor is moved by holding down the appropriate arrow keys, and the location is taken when any alphanumeric key is struck. With the tablet, the pen is moved above the surface of the tablet; the cursor should follow its motion. The location is taken when the pen is pressed to the surface. Either method is referred to as pointing the cursor. If possible you should try to find a tablet; moving the cursor keys can get tedious.

To choose a menu item, point the cursor at a menu box. When using keyboard cursor keys, if the character struck is the first letter of any of the commands, that command is chosen regardless of the cursor position. This helps cut down cursor motion. (It only works when menu input is expected.) The graphics cursor is also used for indicating objects and positions in the mesh window. When a position is required, you can always give numeric data by typing ‘n’ to the graphics prompt. There is also a display option (section 4.2.4.2) which turns on a regularly spaced background grid and rounds input data to the nearest grid point.

Some commands require text input, which cannot be represented with the graphics cursor. In these cases a prompt appears in the top line buffer, and you type input in the same way you would in talking to any other program.

0.2.1.4. Quitting

You can exit **skel** by choosing the ‘quit’ box on the main level menu, and choosing ‘yes’ when it asks to make sure. If you choose ‘no’ you return to the main level without affecting anything. One thing to remember is that most **skel** states are repetitive. For example, if you give a create command, the possible objects you can create will be listed in the operands menu, and you will remain in the create state until you quit it (by choosing the quit box in that menu). Until you quit the create state, items chosen from the main menu are ignored. Furthermore, once you have chosen to create, say, nodes, the program will continually wait for node locations in the mesh window until you move the cursor outside it. Thus in the most general case, you may have to pop up three levels, from the mesh window to the second level menu, from the second menu to the first, and from the first to the outside world. The run light is useful for telling how deep you are.

0.2.2. Objects

A skeleton grid is a collection of three types of object, nodes, edges and regions. Nodes may belong to edges, and edges may belong to regions, so that the hierarchy of objects is

$$\textit{region} \rightarrow \textit{edge} \rightarrow \textit{node}$$

There are commands to create, destroy and manipulate these objects. A fourth object, triangle, is also recognized, but no commands apply to it. The purpose of the program is to create only the outline of a structure; you are not expected to manipulate triangles by hand! In terms of the hierarchy, triangles have links to all three primary types. We examine each of the objects briefly in this section.

0.2.2.1. Nodes

A node is the most primitive object. It has three numbers associated with it, its *x* and *y* coordinates, and its spacing value, to be discussed later (Section 7). Nodes can be created at will, and can be destroyed if they are not on any existing edge. Commands which expect you to choose a node do not require an exact position. The cursor needs only to be closer to the desired node than to any other.

0.2.2.2. Edges

Edges consist of a pair of nodes and a boundary condition code. The boundary condition code has no significance for **skel**, and is passed on through the triangulation phase to be interpreted by whatever simulator reads the grid. PISCES-II interprets this code as an electrode number, SUPREM-IV uses it to choose the boundary condition. The default code is zero. Edges are created and destroyed automatically by commands which alter regions. Edges are displayed as dotted lines, with different line types to distinguish zero and non-zero electrode codes. Commands which expect you to choose an edge do not require an exact position. You need only be closer to the desired edge than to any other. (The perpendicular distance to the edge is used for the comparison.)

0.2.2.3. Regions

A region is the most complex structure. It consists of a material number, which is only of significance to the simulators, and a list of edges. The list must have at least three elements (otherwise, there would be no way to display or pick that region), and is always closed. Since a region does not belong to any higher object, it can be destroyed at will. Regions are not explicitly displayed at present, since the edges which define the region are visible. To choose a region, point the graphics cursor inside the region. A point is defined to lie ‘‘in’’ the region if a line drawn to infinity crosses an odd number of edges, and ‘‘outside’’ that region if it crosses an even number of edges.

0.2.3. Operators

In the following, the notation word1-word2 denotes a command whose name on the first menu is word1 and on the second menu is word2. To invoke any command, point at word1 on the main menu and then word2 on the sub-menu that appears. The main operators available are

0.2.3.1. Create

You may create nodes or regions. To create a node, point at the desired location. It is illegal to create a node on top of another. Nodes created on top of an edge do *not* automatically become part of that edge. The create node command is repetitive, that is, nodes continue to be created until the cursor is moved outside the mesh window. To create a region, pick a series of counterclockwise nodes. The boundary is updated to reflect the addition of each node. If less than two nodes are chosen, the command is aborted without creating any region. It is important to choose the nodes counterclockwise, otherwise the join and split commands will get confused (this problem should go away).

0.2.3.2. Destroy

You may destroy nodes or regions. Point the cursor at the node or inside the appropriate region. The destroy-node command is repetitive (be careful!).

0.2.3.3. Move

You may move a node by picking the node and pointing at its new location. You may move an edge or region by picking the edge or region, then indicating a displacement vector by pointing at a start position and an end position. The edge or region is moved by the difference between the end and start position. You may move the origin by pointing at a node; that node then becomes the origin, and all positions are adjusted. There is also a move-block command which is useful for shifting whole areas of a grid, for instance in generating several MOSFET grids which differ only in their channel lengths. The idea is that you define a section of grid to be stretched. Everything lying outside the section on one side is shifted according to a specified displacement vector, and everything on the other side remains fixed. Nodes lying inside the section are moved by an appropriate fraction. The syntax for doing this is a little convoluted: the section to stretch is defined as the area between two parallel lines, which are perpendicular to and situated at either end of a given line segment. The segment is specified by giving its start and end (*x* and *y* in Fig.1). The displacement vector is then defined by giving a third point (*z*), usually but not necessarily collinear with the first two. Figure 1 might make this more clear.

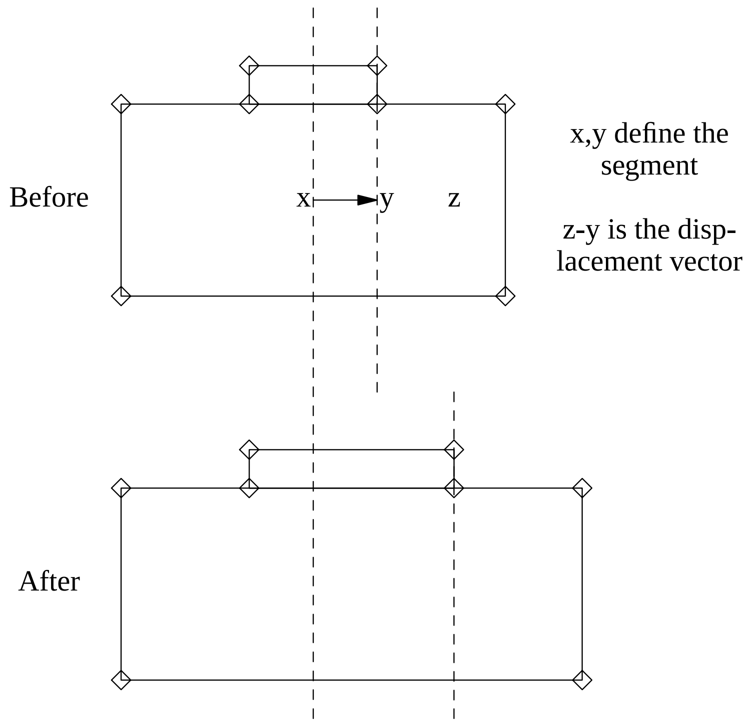

Figure 1 - Using Block Move

0.2.3.4. Alter

The alter command is used to change parameters that are not represented on the display. For nodes, you can modify the spacing value. For edges, you can change the electrode code. For regions, you can change the material number. Input for these commands is typed on the top line. For convenience, all numerical data associated with an object is displayed first, so this can be used to find the exact coordinates of a node.

0.2.3.5. Split

You can split an edge or a region. To split an edge, pick the edge, indicate which end to start adding at, then pick a series of new nodes. For each node added, the edge is split in two. The nodes must already exist. (Figure 2).

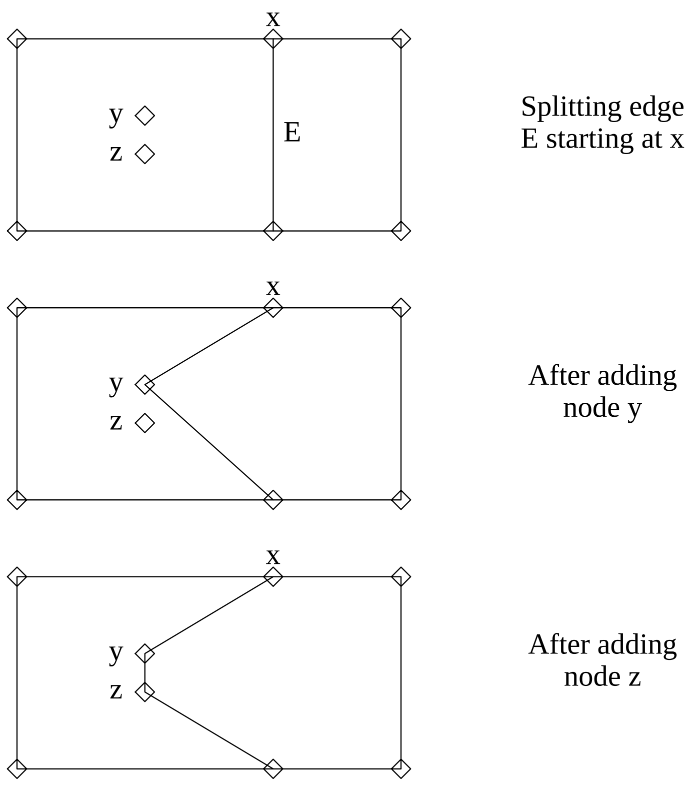

Figure 2 - Splitting an Edge

To split a region, pick the region and two non-adjacent nodes belonging to that region. The region will be split along the line joining those nodes.

0.2.3.6. Join

You can join edges or regions. To join edges, pick two adjacent edges. They will be condensed into one, and the node they previously shared will be free for removal. (Figure 3). Join-edge then waits for another edge to join, and is terminated by moving the cursor outside the graphics window. To join regions, pick two adjacent regions. At present, the regions must be separated by just one edge, as they would be after a split-region command. To join two regions which have an extended interface, first use join-edge to telescope the interface down to one edge.

0.2.4. Other commands

0.2.4.1. Window commands

The window on the mesh may be manipulated in several ways, primarily zooming and panning. The commands are listed below.

0.2.4.1.1. Fresh

The window is simply refreshed. Useful when your screen is messed by an interrupt, or if some command hasn't made all the consequent changes to the display. (This problem should go away).

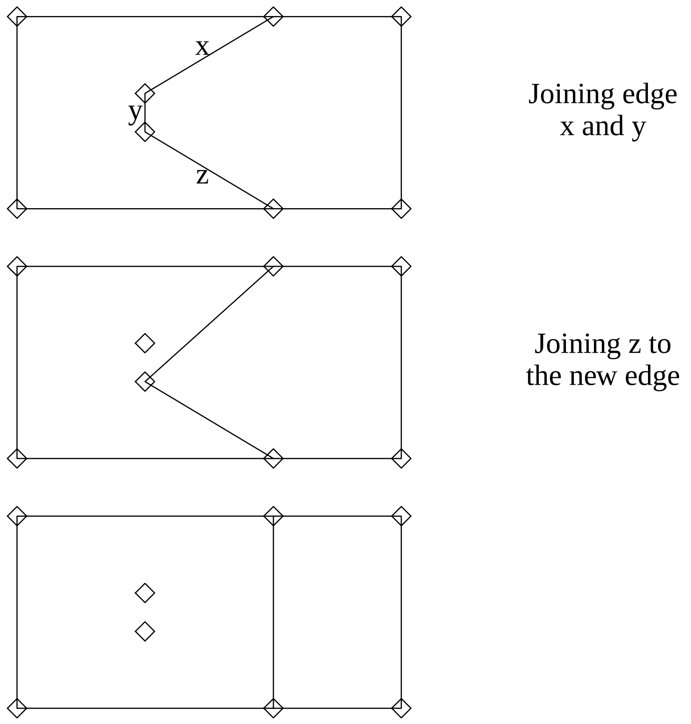

Figure 3 - Joining edges

0.2.4.1.2. In

Zoom in on a part of the grid, specified by pointing at the lower left and upper right of an imaginary box. Window-in and window-out use a stack of windows; window-in creates a new window each time it is invoked, and the previous window is pushed on a stack.

0.2.4.1.3. Out

Pops the stack and redisplays the mesh at the previous size. If you are already at the root of the stack, window-out prompts for the coordinates of a new window in the top line buffer. This is the only way to expand a window to be bigger than the current mesh limits.

0.2.4.1.4. Pan

The window is moved, at the present scale, over the mesh. Window-pan prompts for the start and end of a displacement vector. The window is moved by that vector.

0.2.4.1.5. Reset

The window stack is cleared, and the mesh displayed in a window just large enough to accommodate the entire mesh.

0.2.4.2. Display options

There are a number of options controlling what is displayed on the screen. Some of these are merely to speed up the display on slow terminals (which means 4800 baud in this context), others are useful. All the options are toggles.

Nodes	Display nodes. Initially on. May be turned off to get rid of those diamonds.
Edges	Display edges. Initially on.

Regions	Display region. Initially off. (Not supported at present).
Triangles	Display triangles. Initially off.
Obtuse	Color in obtuse triangles. Initially off.
Stretch	Stretch the mesh to fill the screen. Proportionality is lost. Initially off.
Grid	Set up a background grid to assist in locating the cursor accurately. Initially off. The background grid is usually necessary for precision work. With this option enabled, input to any command that requires positions (create node, move-anything, window-in) is rounded to the nearest point on the background grid. Option-grid prompts for the *x* and *y* grid spacing. If the spacing is so small that the display would be covered with thousands of grid points, the spacing used in the display is rounded up accordingly and you are warned. The spacing used in rounding input data is the specified one, not the one used in the display.
Axis	Display axes. Initially on. Can be turned off to conserve baud.

There is one more display option, given on the command line when the program is invoked. The **-s** option (“short”) abbreviates all text displayed on the screen to one character and makes menu display much faster. Initially you will want to see the full texts, though.

0.2.4.3. File commands

At present there are three file commands in **skel**: file-read, file-write, and file-dplot. File-read creates a text file describing the grid, which can be read back by file-write. Several files can be read in the same run, but you should be careful not to read grids that overlap. File-dplot dumps a file that can be read with the **dplot** system program. The purpose of this option is to allow you to create simple stick pictures using the graphics facilities of **skel**, and make hard copy using **dplot**.

0.2.5. Miscellaneous

0.2.5.1. Error handling/Terminating commands

If you give too little, or invalid, input to a command, **skel** ignores the command, and prints either a warning message or the message “Aborted” in the top line buffer. Any half-completed command can be aborted by simply moving the cursor outside the mesh window and choosing another command. Errors in repetitive commands terminate the repetition.

0.2.5.2. Differences from IGGI

For users familiar with the earlier program IGGI, the following list of differences may help in the transition to **skel**. First, there are new commands to manipulate regions. In particular, you can create a new region. (In IGGI, it was only possible to split and join regions). This has the advantage of being much more flexible, and the disadvantage of introducing the possibility to create overlapping regions. Secondly, the triangulation and drawing programs have been separated, to allow you to run the triangulation in the background. The triangulation algorithm is smarter than that used in IGGI, and usually does not require any user intervention (no more “region densities”). As a result, the triangle type is better described as tolerated rather than supported in **skel**. It is only included to allow you to use the interactive zoom and pan commands to inspect a finished mesh, and to fine-tune triangles by moving nodes around. Another change is the menu handling, where all input is in graphics mode to allow users with tablets to dispense with the keyboard (except for file names).

0.2.6. Interface to Tri.

The output of **skel** is a list of nodes, edges, and regions. PISCES-II requires a triangular grid which covers the entire structure. The triangulation program **tri** performs the mapping from one to the other. **Tri** imposes a particular discipline on its input, in order to generate valid triangulations. It expects a collection of regions that are simple polygons, without any holes or self-crossings. The regions are defined in terms of edges which in turn are defined in terms of nodes, and there are no unconnected edges or nodes. Edges are handled automatically by **skel**, but the user must check whether there are stray nodes before passing the input to **tri**.

Tri begins by computing a local grid spacing (the ‘‘*h*’’ value) for each node, by looking at how closely other nodes approach that node. If a value was specified with **skel**’s alter-node command, that value overrides the computed value. Small values force finer grid. Both user-specified and automatic spacing values are smoothed to avoid large discrepancies between neigboring nodes. The edges are then subdivided in such a way that the spacing at each end of the edge is the value of *h* for the node at that end. The node spacing inside an edge varies geometrically between the two end values. After the edge division phase, the triangulation proceeds by recursively breaking each region into smaller regions, adding new nodes as necessary, until eventually the subregions degenerate into single triangles.

Finally there is an optimization phase, where various manipulations are applied to the grid to make the angles as close to 60° as possible. Depending on the size of the grid and how much optimization is done, this can be quite slow. It was for this reason that the triangulation was separated from the input phase, so that the optimization can be run in the background. See the **tri** manual page for the options available. The triangulated grid is then written to a file that can be read by PISCES-II, or reread with **skel**. **Skel** does not allow any database operations on a triangulated grid; the only valid operators are the move operators, which may be used to fine-tune the angles.

0.2.7. **Skel** usage and idioms

The typical sequence in using **skel** is shown in Figure 4.

Coordinates are entered using either a text editor or directly with the **skel** create-node command. Then the nodes are linked into regions, and the regions are assigned material numbers. After electrodes are defined, the mesh can be saved and triangulated with **tri**. The material numbers have the same meaning as before.

There are a few idioms that may help you in using **skel**. In no particular order, they are...

- Most people using **skel** have an accurate picture of the desired structure, with known coordinates for each point. In this situation, it makes more sense to use a text editor to put the coordinates into a file in ASCII grid format (Appendix I). Then read the text file and use **skel** to set up the edges and regions.
- If you need to create a region where some of the nodes are so close together that they can’t be distinguished on the display, it need not be done in one fell swoop. You can create the region using just the well-spaced nodes, then window in on the smaller features and use split-edge to add the remaining nodes.
- The edge-spacing algorithm uses the spacing at either end to determine the number of subdivisions. Thus a long edge separating two detailed features will be inappropriately subdivided. By adding one node at the center, the grid spacing will increase from one end to the center, and decrease back down to the other end.
- When working with MOSFETs, you may find that the default grid spacing around the thin oxide is too conservative, causing too many nodes to be generated. The number of nodes can be reduced (at the cost of accuracy) by specifying a coarse spacing value for the nodes above and below the oxide.
- To make two copies of an object (for whatever reason), do file-write, then move-block to shift the entire structure somewhere else, and file-read to read in the disk copy (which will be placed in the original position).

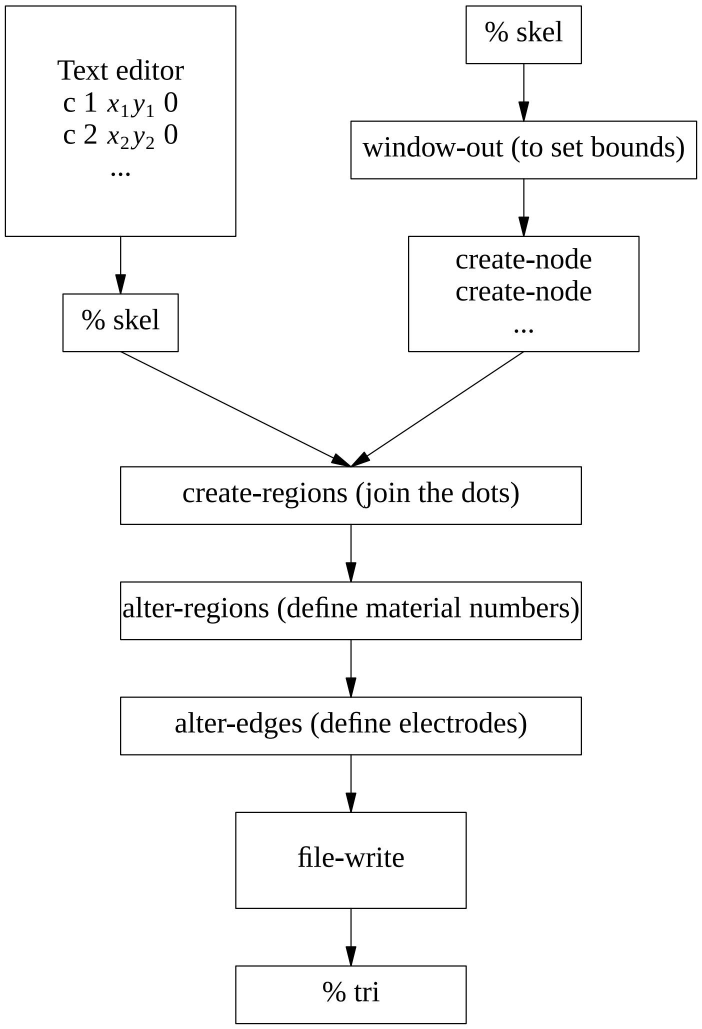
Text editor
c 1 $x_1 y_1$ 0
c 2 $x_2 y_2$ 0
...
% skel
% skel
window-out (to set bounds)
create-node
create-node
...
create-regions (join the dots)
alter-regions (define material numbers)
alter-edges (define electrodes)
file-write
% tri

Figure 4 - Skel Flow Chart

0.2.8. Appendix - Grid format.

The programs dealing with grid all share a common format. Text files are used for a variety of reasons, primarily because they can be sent over simple links from CAD stations to simulation machines, and they can be inspected or created directly by users. The format is a list of nodes, edges, regions and triangles in that order. Both **skel** and **tri** expect regions to be closed and counterclockwise, with no free edges. **Tri** also expects no free nodes. Each object is on a separate line, identified by the first character on that line.

A node is represented as

c *n x y h*

where **c** (‘coordinate’) is the identifying character, *n* is the number of the node, referenced by subsequent edges, and *x*, *y* and *h* are the coordinates and the spacing value of that node. To get the default spacing, give $h = -1$.

An edge is represented as

e *n i j b*

where **e** is the identifying character, *n* is the number of that edge, referenced by subsequent regions, *i*, *j* are the nodes on that edge and *b* is the boundary code of that edge.

A region is represented as

r *n m*

where **r** is the identifying character, *n* is the number of the region, *m* is the material code of that region. This initial card is then followed by a list of the region's edges in the form

b *n*

where *n* is the number of the edge.

A triangle is represented as

t *n r i1 i2 i3 e1 e2 e3*

where **t** is the identifying character, *n* is the number of the triangle, *i1 i2 i3* are the nodes of the triangle, and *e1 e2 e3* are the boundary codes of the triangle sides.

References

[1] R.E. Bank, W.Fichtner *PLTMG User's Guide*, Bell Laboratories Technical Memorandum 82-52111-9