XRLGC USER GUIDE:
A 2-D QUASISTATIC INTERCONNECT MODELING TOOL

BY

CLAIRE E. LESTRADE
French National Engineering Diploma,
Ecole Nationale Superieure des Telecommunications, 1997

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1998

Urbana, Illinois

Second page:

***Certificate of committee approval**

TABLE OF CONTENTS

CHAPTER **PAGE**

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Overview

Constant progress in integrated technology has occurred in recent years, involving extremely high numbers of transistors and interconnects and also improved performance [1]. Signal rise time and cycle time have decreased: consequently, the delay due to interconnects can no longer be neglected.

Interconnections may be found at the chip, package, and board levels, as well as at the assembly level, implying large differences of their density and size [2]. At the chip level, transmission lines connect an increasing number of gates depending on the scale of integration: small (SSI), medium (MSI), large (LSI), or very large (VLSI). The lossy coupled metal interconnects have width and height of about 4 and 3 μm, respectively. At the package level, interconnections link the chip to the board as well as chips to eachother (for multichip modules MCM) using tape-automated bonding (TAB), pin grid array (PGA), ball grid array (BGA) or dual in-line packaging (DIP). The low-loss coupled metal lines are 1-10 mil wide and spaced at about 2-12 mil. Packaging needs to follow adequate guidelines to get short delays, high bandwith, and a large I/O count with a dense wiring and a compact size. Consequently, the high density creates signal integrity problems, including cross-talk, attenuation, distortion, reflections, dispersion, and radiation. The bandwidth limitations are also due to the skin effect, the non-TEM propagation, and propagation of higher order modes and parasitics.

For example, with current technology, both the CPU and the whole cache system may be included on one single microprocessor chip working at a high clock rate (up to 300 Mhz), involving, according to [3], from 4 to 9 million transistors and thus high-density wiring. Those transistors may be interconnected through a structure of between three and five layers. The high density implies also that interconnects have very small width and spacing (less than 1e-6 m), and therefore high resistance (35 to 500 Ω/cm).The interconnects between the processor and the cache including clock lines, control lines, and data lines may be of very long length (up to 1-2 cm). The resulting delay cannot be neglected since the line length is comparable to the signal wavelength (1.8 to 2.5 cm). Moreover, the fast switching speed and high density [1] create increased signal distortion, coupling between transmission lines, and electromagnetic interference (EMI). The use of computer-aided tools [4] for predicting those

losses in signal integrity linked to interconnects has become essential to understand the behavior of high-speed digital circuits. The simple model of capacitive load [3] may not be sufficient anymore for a transmission line of long length and high resistance. The entire distributed *RLGC* model (resistance, inductance, conductance, and capacitance) needs to be considered, involving the full electromagnetic properties of transmission lines. The terminal voltages and currents can be deduced accurately from this model. Moreover, transmission lines may present discontinuities like open-end terminations, bends, junctions, and vias [5], [6], especially in MCMs where the interconnects are often very closely packed.

The prototyping [2] of package interconnections involves the network extraction followed by electromagnetic modeling, determination of the circuit parameters, and lastly the definition of adequate simulation algorithms. The circuit can then be simulated with a software like SPICE or MDS. The modeling methods that can be used to characterize an interconnect are the finite element method (FEM), the partial element equivalent circuit method (PEEC), the finite difference time domain method (FDTD), and the method of moments (MoM). The simulation methods can employ scattering parameters [7], [8], model order reduction, or a difference model. Finally, optimization can be obtained using neural networks, genetic algorithms, or simulated annealing.

The circuit modeling [2] may be static ($d/dt = 0$) or dynamic ($d/dt \leftrightarrow 0$). The static analysis of a physical model considering the integral form of Maxwell's equations implies the use of either Green's function (in spectral domain or in closed form) or the PEEC method, in conjunction with solvers like MoM, the conjugate gradient method (CGM), or the fast multipole method. The charge distribution and the *RLGC* model can be deduced from there. However, if the static analysis considers the differential form of the Maxwell's equations, then the method applied is the static FEM or the Method of Line (MoL), in conjunction with a matrix solver (sparse or full matrix technique). The potential distribution and the *RLGC* model are then deduced. The dynamic analysis may be done in time domain or in frequency domain. In frequency domain, the full-wave techniques yield $E(f)$, $H(f)$, which are the respective Fourier transforms of $E(t)$, $H(t)$. The dynamic circuit model $R(f)$, $L(f)$, $C(f)$, $G(f)$ can then be deduced. This model can be found similarly in time domain once the FDTD or TLM (transmission line method) yields to $E(t)$, $H(t)$.

The transmission line simulation [2] implies consideration of the telegrapher's equations, the eigen analysis (E, H, Λ_m), and determination of propagation parameters (Z_m , Z_C , V_m). Three different methods can then be applied: the transform approach, the Green's

function approach, or the open loop numerical integration method. Proficient transient simulation of transmission lines has been studied in the past [9]-[15]. It has been seen that the optimal approach [10] may be found by considering closely the formulation, the line characterization, the line model, and the transient simulation method used. The resulting optimal method [10] applies the 'time-only formulation' with terminal voltages and currents only, the open loop characterization, the device model (which can be directly incorporated into a circuit simulator as opposed to noncircuit models like scattering parameters), and transient simulation based on indirect numerical integration. This optimal simulation method has been applied to uniform [10] and nonuniform lines [16], [17].

This thesis focuses on the actual modeling of transmission lines through the *R*, *L*, *G*, and *C* parameters. The complex geometries involved in the VLSI system [18] can be modeled in two dimensions as long as the cross-section of the conductors stays constant for a comparatively large distance. The four parameters *R*, *L*, *G*, *C* of quasi-TEM lines may also be computed statically with good accuracy because the transverse distribution of the fields at any time t will be nearly identical to that computed by static analysis. Consequently, XRLGC models the transmission lines statically, using the MoM method in conjunction with the closed-form Green's function. The 2-D quasistatic modeling tool XRLGC can therefore compute accurately the electrical parameters of multilayered, multiconductor structures using those computationally efficient algorithms.

1.2 The 2-D Quasistatic Modeling Tool XRLGC

The XRLGC package models a 2-D multiconductor transmission line in a multilayered dielectric medium by computing the four transmission line parameters *R* (resistance), *L* (inductance), *G* (conductance), and *C* (capacitance), as well as the charge and the potential distributions in this structure. The cross-section of the conductors may be strip-like, rectangular, or circular. It is assumed that the length of the conductors is long compared to the constant cross-section. The microstrip case as well as the stripline case may be handled by using the ground options accordingly (no ground, bottom ground only, or both top and bottom grounds). The output of XRLGC can also be used as an input to another program deducing voltages and currents along the transmission line [2].

The 2-D quasistatic tool XRLGC uses the MoM in conjunction with the closed-form instead of the classical Green's function. The long computation involved in determining the classical Green's function in the spatial domain [18] is thereby avoided. Only one electrostatic

problem is solved in order to compute the capacitance matrix. The inductance matrix *L* is deduced from the equivalent capacitance problem. In addition, the charge distribution obtained during the computation of the capacitance matrix *C* and the inductance matrix *L* is used to deduce the resistance matrix *R* and conductance matrix *G*.

The original RLGC package written by K. S. Oh in 1995 applies the same methods to model a multilayered, multiconductor transmission line. However, RLGC is made of three different programs: Mesh2D, V2D, and RLGC. Mesh2D creates the mesh file for a given conductor, V2D computes the potential distibution, and RLGC computes the *R*, *L*, *G*, and *C* parameters and the charge distribution of the structure considered. Consequently, the design of any structure with the package RLGC implies the multiple use of these three different programs. For example, Mesh2D needs to be called for each conductor of the structure. Similarly, a complex input file needs to be completed for each structure considered, containing the detailed geometry and characteristic of each element, layer or conductor. The use of the package RLGC is therefore difficult and time consuming.

However, the program XRLGC presented in this thesis links the three programs together using a graphical user interface (GUI). This GUI has one main goal: to provide the user a way to use easily the very sophisticated modeling tool described above. The design [19] of GUIs for electromagnetic simulation tools results not only in user-friendliness, but also involves a significant increase in user productivity. First, the input data files don't need to be typed by the user. They are automatically extracted from the drafting tool, which results in important time savings. The simulation is also usefully documented by the precise drawing involved in the design of the multilayered, multiconductor structure considered. Finally, the GUI allows the user to interactively run simulations and analyze the simulation results without concern for lower-level computer processing. XRLGC can therefore be used without any computer expertise.

1.3 Structure of the Thesis

The ease of learning a GUI-based program makes it more accessible to new users. However, the accuracy of the results obtained through the interface is fixed by the simulation tool itself. In other words, the description of a GUI would be incomplete [19] without a detailed portrait of the electromagnetic tool involved, including capacities, limits, and reliability. Consequently, before getting deeper into the use of the graphical user interface itself, Chapter 2 will briefly present the methods implemented by K. S. Oh in his 2-D

quasistatic modeling tool RLGC.

Among other important criteria [19], the GUI should be easy to use; it should comply to industry standards like X-Windows, thus escaping any hardware or software needs and/or compatibility problems; and it should involve online help on useful topics. The interface presented here possesses all these features. Nevertheless, without a good user manual, the user may occasionally be unsure of various technical details. Chapter 3 gives a quick overview of the essential topics, including the installation of XRLGC, the online help, the various menu options, and directions for the design of any interconnect structure. These guidelines may then be applied using the examples given in Chapter 4.

Because integrated circuit technology develops and changes rapidly, computer aided design (CAD) tools need to be capable of adapting quickly and efficiently to the future needs of the user. The comments and notes embedded in the source code might not always be sufficient to explain the whole program structure so that it can be proficiently improved. Consequently, Chapter 5 documents in detail the structure of the program XRLGC so that a C-developer can modify the source code adequately.

CHAPTER 2
BACKGROUND

2.1 Methods Implemented

The interconnects considered by XRLGC have a complex 2-D structure involving up to 20 layers and up to 20 conductors of small cross-section compared to their length. The cross-section shapes available for the conductor in XRLGC are the strip, the rectangle, and the circle. The limits given to the number of layers and conductors implied in the structure may be easily changed in the resource file. The XRLGC program may also be extended to any polygon-shaped conductor (cf. Section 5).

The *RLGC* parameters of an interconnect are computed accurately in the XRLGC program by using the following methods [18], [20]. Capacitance *C* is determined using the MoM associated with the closed-form Green's function, avoiding the use of numerical integration or nested infinite sum involved in the classical, full Green's function. Inductance *L* is deduced from the equivalent capacitance problem. The charge distribution obtained during the computation of *C* and *L* is used to deduce resistance *R* and conductance *G*. In other words, no additional electrostatic problem is solved in the process. The model of the diagonal resistance matrix has been chosen because it is not as sensitive to the choice of the current excitation as is the usual nondiagonal resistance matrix. Both losses on conductors and ground planes are taken into account in the computation of *R*. MoM will be briefly explained, and its application to the capacitance computation presented. The methods used for the computation of the three other matrices *L*, *G*, and *R* will then be introduced succinctly. Finally, the closed-form version of the Green's function used in MoM will be defined in the last section.

2.2 Method of Moments

2.2.1 Basics

MoM [2] basically consists of solving

$$L\,\acute{a}f\acute{e}=g \tag{1}$$

where *L* is an integral or differential operation, *f* is an unknown function, and *g* is a known function. We will see how this problem exactly fits the electrostatic field problem, where *L* is the Green's function *G*, *f* is the charge distribution *p*, and *g* is the potential ϕ.

The first step in MoM involves expanding the function *f* in a set of basis functions *Bn*, which leads to

$$f = \sum_n \alpha_n B_n \tag{2a}$$

$$\sum_n \alpha_n L(B_n) = g \tag{2b}$$

In addition, the inner product of Equation (2b) with a set of test functions T_m develops into

$$\sum_n \alpha_n \langle T_m, L(B_n) \rangle = \langle T_m, g \rangle \tag{3}$$

Equation (3) may also be understood as the matrix equation

$$[l_{mn}][\alpha_n] = [g_m] \tag{4a}$$

where l_{mn} and g_{mn} are defined by

$$l_{mn} = \langle T_m, L(B_n) \rangle \tag{4b}$$

$$g_m = \langle T_m, g \rangle \tag{4c}$$

The unknown parameters [α_m] may be obtained by inverting the known [l_{mn}] matrix and taking its product with the known vector [g_m]. However, the computationally expensive matrix inversion may be avoided by using alternate methods [21].

Moreover, both the matrix [l_{mn}] and the vector [g_m] depend of the sets of basis and test functions, which in general are chosen to be equal. Examples of commonly used functions are given in $x_{n+1} - x_n = \Delta$

$$B_n(x) = \begin{cases} 1 & x_n - \frac{\Delta}{2} < x < x_n + \frac{\Delta}{2} \\ 0 & \text{Otherwise} \end{cases} \tag{5a}$$

$$B_n(x) = \begin{cases} 1 - |x| & x_n - \Delta < x < x_n + \Delta \\ 0 & \text{Otherwise} \end{cases} \quad (5b)$$

where (x_n) is an array of real number. Once the sets of basis and test functions have been chosen, the matrix Equation (4a) may be solved, and the unknown function *f* is easily deduced from Equation (2a).

2.2.2 Application to a simple electrostatic field problem

If we apply MoM to a 1-D electrostatic field problem [2], we get

$$\varepsilon \nabla D = q \quad (6a)$$

$$E = -\varepsilon \nabla \Phi \quad (6b)$$

which leads to

$$\varepsilon^2 \nabla \Phi = -\frac{q}{\varepsilon} \quad (6c)$$

$$\Phi = \int G(x|x')q(x')dx' \quad (6d)$$

In Equation (6d), the Green's function $G(x,x')$ and total potential ϕ are known, while the charge distribution $q(x')$ is unknown. This is equivalent to Equation (1), where L is the Green's function G, f is the charge distribution q, and g is the potential ϕ.

2.3 Capacitance Matrix

For the sake of simplicity, the MoM documented in Section 2.2 was only applied to 1-D electrostatic field problems, the unknown charge distribution being only a function of x. If the charge distribution considered was two or three dimensional, the principle would stay the same. For instance, in the 2-D domain [18], the relationship between the potential $\phi(r)$ applied on a conductor and the charge $q(\rho)$ accumulating on the surface of the conductor is given by

$$\phi(r) = \int_{\Omega} G^{2D}(\rho|\rho')q(\rho')dr' = \langle G^{2D}, q \rangle \quad (7)$$

where Ω is the contour of all the conductors, ground excluded, and G is the closed-form Green's function. Consequently, the generic form of integrations involved in the construction of the moment matrix for the 2-D electrostatic problem is

$$\int_{\rho_t}\int_{\rho_s} T(\rho)\, G^{2D}(\rho|\rho_s)\, B(\rho_s)\, d\rho_s\, d\rho \tag{8}$$

The functions T and B are the testing and basis functions which have been seen in the above 1-D MoM. The segments l_S and l_t correspond to the source line and the testing line, respectively. The double integration is then reduced to a single integration using the method of collocation with pulse-type basis functions and a testing point p_C located at the center of the testing segment:

$$\int_{\rho_s} G^{2D}(\rho_c|\rho_s)\, d\rho_s \tag{9}$$

Now, according to [18], if we apply MoM to the integral equation (7) by approximating each conductor with a polygon and by expanding the charge density in a set of basis functions, we obtain

$$\begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_{N_c} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M}_{1,1} & \boldsymbol{M}_{1,2} & \cdots & \boldsymbol{M}_{1,N_c} \\ \boldsymbol{M}_{2,1} & \boldsymbol{M}_{2,2} & \cdots & \boldsymbol{M}_{2,N_c} \\ & & & \\ \boldsymbol{M}_{N_c,1} & \boldsymbol{M}_{N_c,2} & \cdots & \boldsymbol{M}_{N_c,N_c} \end{bmatrix} \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_{N_c} \end{bmatrix}$$

(10a)

where:

$$\mathbf{V}_i = \left[V_i \;\; \cdots \;\; V_i \right]^T \quad \text{length } N_i$$

$$\mathbf{q}_i = \left[q_1^i \;\; \cdots \;\; q_{N_i}^i \right]^T \quad \text{length } N_i$$

$$\left[\boldsymbol{M}_{i,j}\right]_{p,q} = \int_{I_q^j} G^{2D}(\rho_c^{i,p}|\rho_s)\, d\rho_s \quad \text{size } N_i \times N_j$$

(10b)

Here, N_C is the number of conductors considered in the structure. For a given conductor i, N_i is its number of basis functions, V_i is its voltage relative to the ground, q^i_j is the coefficient of q^i according to the j^{th} basis function, I^i_q corresponds to the q^{th} line segment, and $\rho^{i,j}_C$ is the center point of the j^{th} basis function used in the point matching

technique.

The charge distribution can then be deduced by solving Equation (10). Consequently, the total charge Q_i of the conductor *i* may be expressed as a function of the length and the charge density of each line segment:

$$Q_i = \sum_{j=1}^{N_i} l_j^i q_j^i \tag{11}$$

The capacitance matrix is then deduced immediately by solving

$$\boldsymbol{C}\begin{bmatrix} V_1 \\ \mathrm{C} \\ V_{N_c} \end{bmatrix} = \begin{bmatrix} Q_1 \\ \mathrm{C} \\ Q_{N_c} \end{bmatrix} \tag{12}$$

where V_i is the voltage of the conductor *i* relative to the ground.

2.4 Inductance Matrix

The MoM applied to a 2-D electrostatic problem gave in Section 2.3 the charge distribution $q(\rho)$, from which was deduced the total charge Q_i for each conductor and, therefore, the capacitance matrix *C*. According to [18], the solution of 2-D magnetostatic problems (Equation (13)) can often be found by considering the equivalent electrostatic problem with $V = \Psi$, where Ψ_i is the magnetic flux difference between the i^{th} signal conductor and the reference conductor.

$$L = \Psi I^{-1} \tag{13}$$

If Q_{eq} and C_{eq} are the charge and capacitance matrices found in (11) and (12) as a solution to the equivalent electrostatic problem defined above, the inductance matrix *L* can easily be computed by solving

$$\begin{aligned} I &= c^2 Q_{eq} \\ L &= \frac{1}{c^2} C_{eq}^{-1} \end{aligned} \tag{14}$$

2.5 Conductance Matrix

Because conductance matrix *G* models losses due to the dielectric, it can be computed by applying *N* voltage excitations to the multilayered, multiconductor system, where *N* is the number of modes, and by observing the resulting shunt current [18].

If we consider *N* independent voltage excitations V_j and the corresponding matrix *V*, the conductance matrix *G* can be computed by solving

$$\boldsymbol{G} = \boldsymbol{I}^{s}\boldsymbol{V}^{-1}$$
$$I^{s}_{i,j} = \int_{c_i} q^{j}(\rho')\,\sigma(\rho')/\varepsilon(\rho')\,d\rho' \qquad (15)$$

where $I_{i,j}$ is the shunt current created by V_j on the i^{th} conductor.

The charge distribution q^i (ρ') of the lossy system can be approximated by the charge distribution of a lossless system defined in part 2.3; therefore, no additional electrostatic problem needs to be solved. The shunt current $I_{i,j}$ may be deduced from the charge distribution, and Equation (15) leads immediately to the conductance matrix *G*.

2.6 Resistance Matrix

For low frequencies, the diagonal resistance matrix is obtained simply by considering the inverse of the conductivity multiplied by the cross-section of each conductor. However, both edge and proximity effect appear at high frequency, resulting in a nonuniform current distribution on each conductor and in an increased complexity for the resistance matrix *R* [18].

Nevertheless, even for high frequency, the resistance matrix is assumed to be diagonal [18]. This diagonal matrix does not model power losses as accurately as the nondiagonal matrix, but it is less sensitive to the current distribution considered and therefore more adapted to the calculation of the resistance matrix *R*. Losses due to the ground planes as well as to the conductors are taken into account in this matrix.

If we consider the total power loss P_i due to the serie of conduction current vectors

I_i^c in the multiconductor structure, the diagonal resitance matrix R is given by

$$\left[P_i^c\right]^t R \left[P_i^c\right] = P_i \tag{16}$$

where $R_{s,j}$ is the surface resistivity of the j^{th} conductor and the conduction vector and power loss are given by

$$\begin{aligned}
\left[P_i^c\right]_j &= \int_{c_j} J_i^c(\rho')\, d\rho' \\
P_i &= \sum_{j=1}^{N} \int_{c_j} R_{s,j} \left[J_i^c(\rho')\right]^2 d\rho' \\
R_{s,j} &= \sqrt{\frac{\pi f \mu}{\sigma_j}}
\end{aligned} \tag{17}$$

Once the power loss and the conduction current vector are known, the diagonal resistance matrix R may then be deduced from Equation (16). Nevertheless, according to [18], this last equation may still be simplified by noting that R is a diagonal matrix and that it may be replaced by a vector containing the diagonal element of R only.

2.7 Closed-Form Green's Function

The program XRLGC uses the closed-form Green's function [18] instead of the traditional Green's function, in order to avoid the numerical integration of a nested infinite sum involved in using an integral equation with the classical Green's function [22]-[25]. An additional simplification is performed by considering real instead of complex exponential functions [26] in order to approximate the real Green's function in the spectral domain. If the Green's function is replaced by its closed-form version in Equation (8) in Section 2.3, we get the closed-form integration formula used in XRLGC for the construction of the moment matrix.

The closed-form Green's function for a multilayered dielectric medium is obtained by approximating the spectral-domain Green's function with real exponential functions, and by inverting it analytically to the space domain. Because MoM is already an approximate way of solving an integral equation, it is not necessary to have perfect precision for the Green's

function *G*, and a moderate number of exponential functions (5 to 7) was generally sufficient to solve most problems. The method used by K. S. Oh [18] to get the expression of the closed-form Green's function is summarized below.

As shown in Fig. 2.1, the coordinates (x_o ,y_o ,z_o) indicate the position of the source in the m^{th} layer, and the coordinates (*x*,*y*,*z*) indicate the point of computation of the Green's function in the n^{th} layer. The length d_n corresponds to the distance between the top of the n^{th} layer and the optional bottom ground plane, and ε_n corresponds to the permittivity of the n^{th} layer.

We will first consider the simple case involving a multilayered structure with no ground or one bottom ground only. In this case, the exponential approximation may be directly applied to the spectral domain expression of the Green's function, leading to

$$\tilde{G}(\gamma,y|r_o)=\frac{1}{2\varepsilon_m\gamma}\left[K_1^+(\gamma,m,n)e^{\gamma(y+y_o-2d_n)}+K_2^+(\gamma,m,n)e^{\gamma(y-y_o+2(d_{m-1}-d_n))}\right]$$
$$+\frac{1}{2\varepsilon_m\gamma}\left[K_3^+(\gamma,m,n)e^{\gamma(-y+y_o)}+K_4^+(\gamma,m,n)e^{\gamma(-y-y_o+2d_{m-1})}\right] \quad (18)$$

for $y > y_o$. The expression found for the case $y < y_o$ is similar.

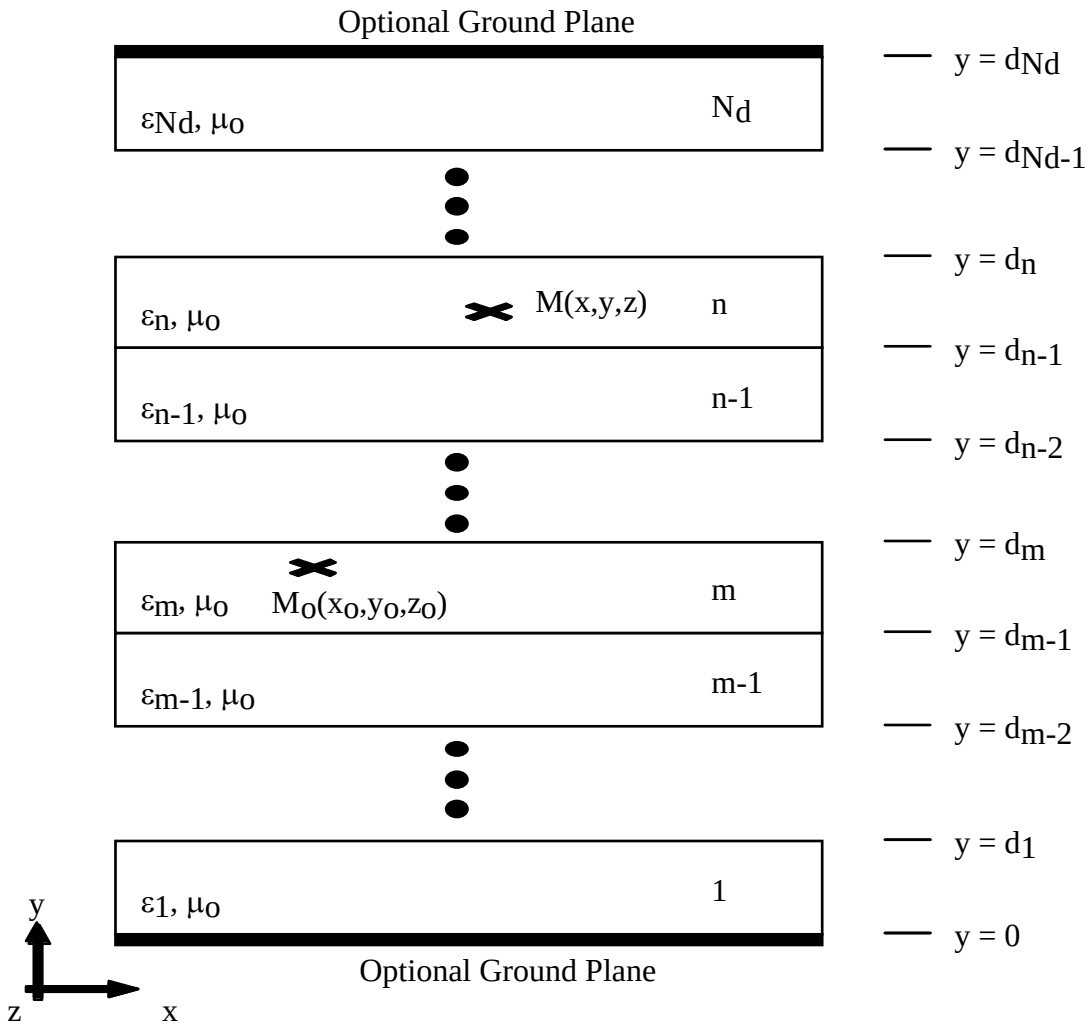

Fig 2.1: Multilayered dielectric medium

The four coefficient functions K_i, i = 1 to 4 (m,n,γ) of the spectral-domain Green's function can be approximated by the exponential form given in

$$K_i^{''}(m,n,\gamma) = \sum_{j=1}^{N_{m,n,i}} C_{m,n,i}^{'',j} e^{a_{m,n,i}^{'',j}\gamma}$$

(19)

where m and n correspond to the layer where the source is located and to the layer where G^{2D} is being evaluated, respectively, and $N_{m,n,i}$ is the number of exponential functions used in the approximation. For more details concerning the evaluation of the coefficient functions, refer to [18].

Once Equation (18) of the spectral-domain Green's function is analytically inverted into the space domain, the final expression obtained for the closed-form Green's function is given by

$$G^{2D}(\rho|\rho_o) = -\frac{1}{2\Pi\varepsilon_m}\sum_{i=1}^{4} f_i^{2D,''}(\rho|\rho_o)$$

(20a)

where for i = 1,

$$f_1^{2D,+}(\rho|\rho_o) = \sum_{j=1}^{N_{m,n,1}^{+}} C_{m,n,1}^{+,j} \ln\left(\sqrt{(x-x_o)^2 + (y+y_o-d_n+a_{m,n,1}^{+,j})^2}\right)$$

(20b)

Similar expressions for the functions $f_i^{+/-}$ with i = 2, 3, and 4 may be found easily.

When there is both top and bottom ground, the expression for the spectral-domain Green's function will be similar except that the four coefficient functions $K_i^{+/-}$ (γ,m,n) have a pole at γ = 0 that must be extracted before the approximation through real exponential functions [18]. The spectral-domain Green's function may hence be written as the sum of two terms:

$$\tilde{G}(\gamma,y|r_o) = R_{m,n}\tilde{G}^{h}(\gamma,y|r_o) + \tilde{G}'(\gamma,y|r_o)$$

(21a)

where G' corresponds to the case of no ground or only one bottom ground, which has been seen previously, and G^h corresponds to an homogeneous medium between the top and bottom grounds. According to [18], G^h is given by

$$\tilde{G}^{2D,h}(\rho|\rho_o) = \frac{1}{2\pi}\sum_{k=-4}^{+4} \ln\left[\frac{\sqrt{(x-x_o)^2+(y-y_o-2kh)^2}}{\sqrt{(x-x_o)^2+(y+y_o-2kh)^2}}\right]$$

(21b)

The closed-form Green's function has therefore been evaluated for all cases involving no ground, only one bottom ground, or both bottom and top grounds.

CHAPTER 3
OVERVIEW OF XRLGC

This overview will help the user to manipulate the program XRLGC easily and efficiently. The first section will document the installation of XRLGC . The second section will then describe the use of the online help dialog. The third section will go into more detail about the different options given by the menu, including file managing in the file menu, units, quality and scale options in the edit menu, and finally displaying the results through the view menu. The design of a multilayered, multiconductor structure with the help of the interface will also be studied in the last section.

3.1 Compiling and Running XRLGC

The program XRLGC may be compiled by following the subsequent instructions. First, in order to **unzip** the file, the user needs to type

>gzip -d XRLGC.tar.gz
>tar -xvf XRLGC.tar
or:
>tar -zxvf XRLGC.tar.gz.

Second, in order to **compile** the program on an HP workstation, the user will type these additional commands.

>make -f Makefile_HP
or:
>cp Makefile_HP Makefile
>make

Third, in order to **run** the program, the user may simply type

>XRLGC&.

3.2 Using the Help Dialog

Once the program XRLGC is successfully compiled and running, the next important

step is to look at the help menu, which gives the option to use either an index or a context sensitive help dialog. In order to use the latter, the user may press the ‘Context Sensitive Help’ option of the help menu. The mouse arrow will take the shape of a question mark. The user may then use the mouse to click on the desired element (a pushbutton, a menu option, etc.). The help dialog will open and tell the user more about this particular element.

In Fig. 3.1 one can see for example some instructions about the command ‘Exit’: according to the help dialog, the option ‘Exit’ in the file menu allows the user to quit the program. The help dialog also advises the user to save every modification in the circuit before using the command. Such information is available on every option available through the menu or through any other element (action buttons like ‘Clear,’ ‘Plot,’ ‘New Conductor,’ the drawing area itself, and more). When the user is done reading the corresponding instructions, the ‘dismiss’ button will close the help window.

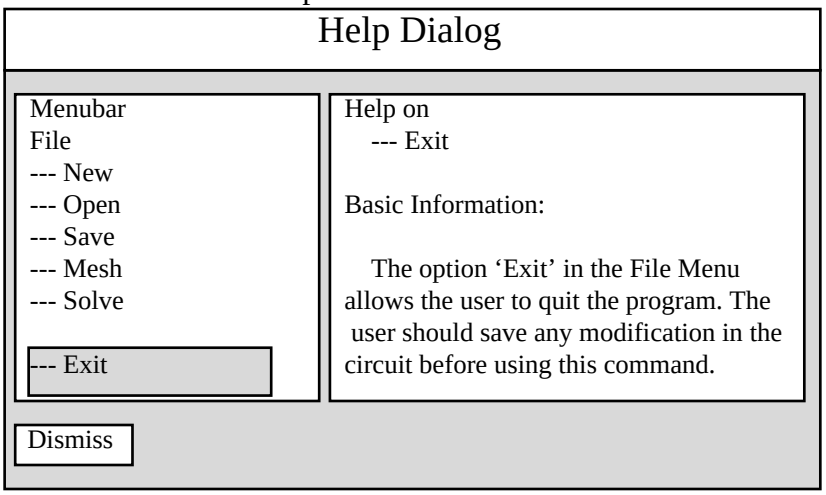

Fig. 3.1: Help dialog

In order to use the index, the ‘Using Help’ option of the help menu should be selected. The same window as for the ‘Context Sensitive Help’ will appear on the screen. The user just needs to choose the desired element in the left-hand list. The corresponding information should appear on the right-hand side of the dialog.

3.3 Using the Menu

3.3.1 Introduction

After getting more familiar with the XRLGC program and the different options

through the help dialog described above, the user may try to look at the menubar located at the top of the GUI. The menubar shown on Fig. 3.2 includes the file, edit, view, and help menus. The help menu has already been seen in detail in the preceding section, so we will concentrate here on the three first menus: file, edit, and view.

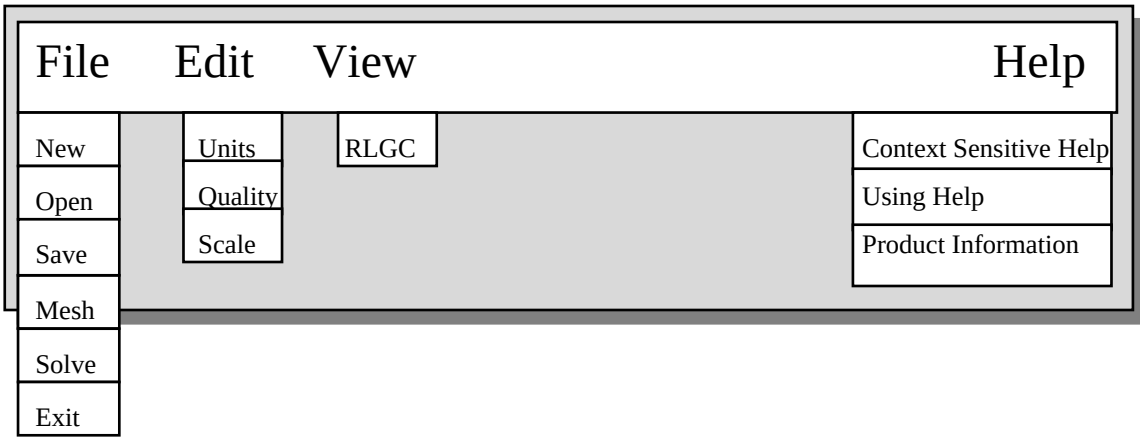

Fig. 3.2 : XRLGC menubar

3.3.2 File menu

The file menu allows the user to create, open, or save a file, or to run a simulation on the data implied. The option 'New' in the file menu allows the user to create a new circuit. For a circuit called 'Example,' five files may be created using options 'Save' and 'Solve': 'Example' contains the structure of the circuit (layers and transmission lines); 'Example.in' contains the input file for RLGC; and 'Example.out,' 'Example.Q2D,' and 'Example.V2D' contain the result of the solving process (*RLGC* matrix, Charge and Voltage distribution). Moreover, mesh files containing the mesh for each conductor inside of the structure may be created. In order to simplify the notations, this paper will always use the name 'Example' for the circuit considered, unless otherwise indicated.

The option 'Open' in the file menu allows the user to open a circuit that was originally created with the program XRLGC. Only the file 'Example' is needed to open a circuit. If this circuit has already been simulated, the files 'Example.out,' 'Example.Q2D,' and 'Example.V2D' will also be available to the program, and the user may view the corresponding *RLGC* matrices through the menu 'View-RLGC.' If desired, the circuit can be simulated again with other options, increasing for example the number of basis functions and therefore the quality of the simulation, or simply by modifying the multilayered,

multiconductor structure.

The option ‘Save’ in the file menu allows one to save the circuit currently being worked on in the ‘Example’ format. This command can only be used after having either created a new file or opened an already-existing file. The user does not need to save the circuit displayed on the drawing area except to keep it for future use: once the circuit is saved, it may be opened and modified any time using the option ‘Open’ in the file menu.

The option ‘Mesh’ in the file menu enables the user to create the mesh files for all the conductors of the circuit. However, the number of basis functions per side has to be defined first in the menu ‘Edit-Quality’ (cf. Section 3.4.2). Once the quality of the simulation has been defined, the user may start the mesh creation using the ‘Mesh’ option in the file menu. An information window will warn the user when the mesh creation is done. There are three types of mesh files: strip.mesh, rect.mesh, and circle.mesh, depending on the shape of the conductor. One file will be created for each conductor involved in the structure. For example, if the structure includes two rectangular conductors and one circular conductor, the files created will be ‘rect1.mesh,’ ‘rect2.mesh,’ and ‘circle3.mesh.’

The option ‘Solve’ in the file menu allows the user to run the simulation on the circuit. It will use as an input the circuit displayed in the drawing area and the mesh files created by the option ‘Mesh.’ Consequently, the mesh creation should be done before running the simulation with ‘Solve.’ Once the option ‘Solve’ has been chosen, the program XRLGC will start to compute the *R, L, G,* and *C* parameters of the structure considered. As soon as the *RLGC* simulation is done, a new custom dialog will appear on the screen. This dialog allows the user to specify the window used for the potential simulation. The user will also decide which conductor should be excited, and the number of points that should be used in the x and y directions. Having typed in the corresponding information, the user may click on the button ‘OK’ of the potential simulation dialog. This allows the program XRLGC to complete the full simulation. However, if instead the button ‘Cancel’ is chosen, then the potential distribution will not be computed. Once the simulation is done, an information window will warn the user that the circuit has been solved. Charge and potential distribution will then be available in the files ‘Example.Q2D’ and ‘Example.V2D.’ The *RLGC* matrix will be available in the file ‘Example.out,’ but it may also be displayed using the command ‘View-RLGC’ on the menu.

The option ‘Exit’ in the file menu allows the user to quit the program. The user should save any modification in the circuit using the option ‘File-Save’ before using this command.

Any modification of the circuit that has not been solved using ‘File-Save’ will be lost once the user exits the application.

3.3.3 Edit menu

The edit menu enables the user to modify some important parameters, like the units used as reference, the scale of the drawing area, and the precision of the simulation. The option ‘Units’ in the edit menu allows the user to modify the units used as reference concerning length, capacitance, inductance, and resistance. The default length units are millimeters. If the length unit is modified, absolutely all the distances used in the circuit will be changed to those new units—the distance defined for the structure before as well as after this modification. Obviously, the circuit displayed on the drawing area will not change because proportions are maintained.

The option ‘Quality’ in the edit menu allows the user to modify the speed of the simulation and therefore its precision. The user determines the number of basis functions used for the mesh construction. This number is valid for all the conductors involved in the structure. The minimum and maximum numbers that may be considered are five and one hundred basis functions per side, the default value being five basis functions per side. This number defines the quality and speed of the simulation. The higher the number, the slower and more precise the simulation. The lower the number, the quicker and more imprecise the simulation. Obviously, this choice should be done before the mesh is created.

The option ‘Scale’ in the edit menu allows the user to zoom in on the lower left part of the drawing area by choosing the size of the window. The default value $X_{max} = Y_{max} = 100$ (default unit: mm) may be modified through the scale dialog.

3.3.4 View menu

The view menu enables the user to display the result of the simulation. The option ‘RLGC’ in the view menu opens a window that displays the *RLGC* matrix of the circuit. This menu may be improved by adding two options: charge and potential distribution. For now, those results are only available through the ‘Example.Q2D’ and ‘Example.V2D’ files.

3.4 Design of a Multilayered, Multiconductor Structure

3.4.1 Introduction

Now that each option of the menu has been clearly identified, it is time to see how to use XRLGC to actually design a multilayered, multiconductor structure. We will consider the different design options below, then see actual examples in Section 4.

The interface is divided into four main parts: the menu on the top part, the drawing area on the right and two adjacent columns on the left part. The first column on the left, called the 'layer' column, is used to design the dielectric layers of the structure considered. The second column, called the 'conductor' column, is used to add the conductors to this structure. These two columns allow the user to specify diverse characteristics for both layers and conductors: shape, as well as material, size, and position. They also allow the user to create new conductors and layers, to display or 'plot' them, to erase them, or to modify them.

The layer and conductor columns on the left of the interface have a very similar structure, including analog options and design techniques. Multiple buttons—like 'Plot,' 'Erase,' '<—,' or '—>'—are included in both columns, and they act the same in both columns, except that the buttons on the layer column deal only with dielectric layers while the buttons on the conductor column deal only with the conductors. There may be some slight differences between them, which will be explained in detail in the paragraphs below. We may now consider in more detail all the different options that will allow the user to design the final multilayered, multiconductor structure.

3.4.2 Design options

The option 'New Conductor' located on the conductor column allows the user to create a new conductor of the shape chosen beforehand (strip, rectangle, or circle). 'Strip' is the default value. Once the user has created this new conductor, the right characteristics can be inserted in the proper places (width, height, and origin (x,y)). The conductor can then be displayed using the pushbutton 'Plot.' The user can also draw with the mouse directly on the drawing area. Similarly, the option 'New Layer' located on the layer column allows the user to create a new layer. Afterwards, the user should insert the corresponding characteristics in the layer column (width, height, origin (x,y), relative permittivity and resistivity). The dielectric will be displayed on the drawing area using the Pushbutton 'Plot.' The user can also draw with the mouse directly on the drawing area. However, the different dielectrics will always lie on top of each other.

Once the conductor (or the layer) is displayed on the screen, it is always possible to modify it again by correcting the corresponding data on the interface (dimensions, shape, origin, or other characteristics) and pressing ‘Plot’ again. However, if other conductors (or layers) have been created using ‘New Conductor’ (or ‘New Layer’), it is possible to go back to this first conductor (or layer) by using the arrows, and then correct the desired data as described above. The corrected conductor (or layer) will be displayed by pressing ‘Plot’ once again. In other words, the user has permanent access to the whole structure. The left arrow ‘<—’ on the conductor column will allow the user to go back to the conductors of the same shape (strip, rectangular, or circular) that have already been displayed and to modify their main characteristics including their width, height, and location (x,y). Similarly, the pushbutton ‘<—’ on the layer column allows the user to go back to already-created layers and to modify their characteristics (width, height, location (x,y), resistivity, and permittivity). The right arrow ‘—>’ allows the user to go forward again, assuming that the pushbutton backward ‘<—’ has already been used.

For conductors, the combination of these two functions in the conductor column (‘—>’ and ‘<—’) allows the user to go back and forth between the different conductors of the same shape (strip, rectangular or circular) and to modify their characteristics (width, height, and location (x,y)). Similarly, the combination of these two functions on the layer column allows the user to go back and forth between the different layers displayed and to modify their characteristics (width, height, location (x,y), resistivity, and permittivity).

The pushbutton ‘Plot’ allows the user to display the conductors/layers in the drawing area. Once the characteristics of a conductor/layer have been modified using the left and right arrows, the user may push ‘Plot’ to see those modifications on the screen. The pushbutton ‘Erase’ allows the user to erase one conductor (or layer) in the drawing area. The user should use the left and right arrows to select the right conductor (or layer).

3.4.3 Using the mouse

The Drawing area will display layers and conductors of different shapes (strip, rectangular or circular) according to the wishes of the user. Two methods may be used: (1) The conductors and the layers may be drawn directly on the screen with the mouse (press ‘New Conductor’ or ‘New Layer’ to go back and forth between both, and choose the shape of

the conductor as desired). (2) They can also be automatically drawn according to the parameters given by the user on the left of the drawing area. After choosing width, height, and position of the element, just press ‘Plot.’

CHAPTER 4
EXAMPLES OF APPLICATION

The third chapter presented a quick overview of XRLGC. Those guidelines may now be successfully applied with a few hands-on examples given in this chapter. We assume in these examples that the program has already been compiled according to the instructions in Section 3.2. Note that this interface saves more time as the structure considered gets more complex. In the first example we will consider a very simple structure: a single layer with a single microstrip line. In the second example we will consider a slightly more complex structure involving two layers and three identical rectangular conductors. Finally, the third example shows useful options including file management and modification of a given structure.

4.1 Starting XRLGC

Once the program is compiling, XRLGC may be started by typing in the command line

> XRLGC&.

A small window should appear on the screen, describing succinctly the program (see Fig. 4.1). Once the information is displayed, click ‘OK’ to get to the main program. The main window will then appear on the screen (see Fig. 4.2). Note the presence of a drawing area on the right-hand side, where the simple structure will be drawn. The left part of the interface is divided into the layer column and the conductor column, as shown in Chapter 3. Those two columns are essential to the design of the multilayered, multiconductor structure.

4.2 A Single Microstrip Line

4.2.1 Hypothesis and geometry considered

We consider the structure shown in Fig. 4.3. It is assumed that the line is infinitely thin and the number of basis function is 50.

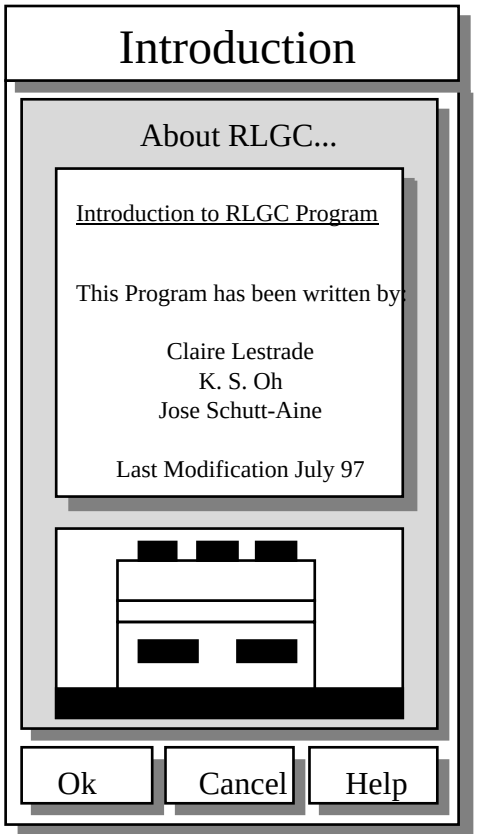

Fig 4.1: Introduction window

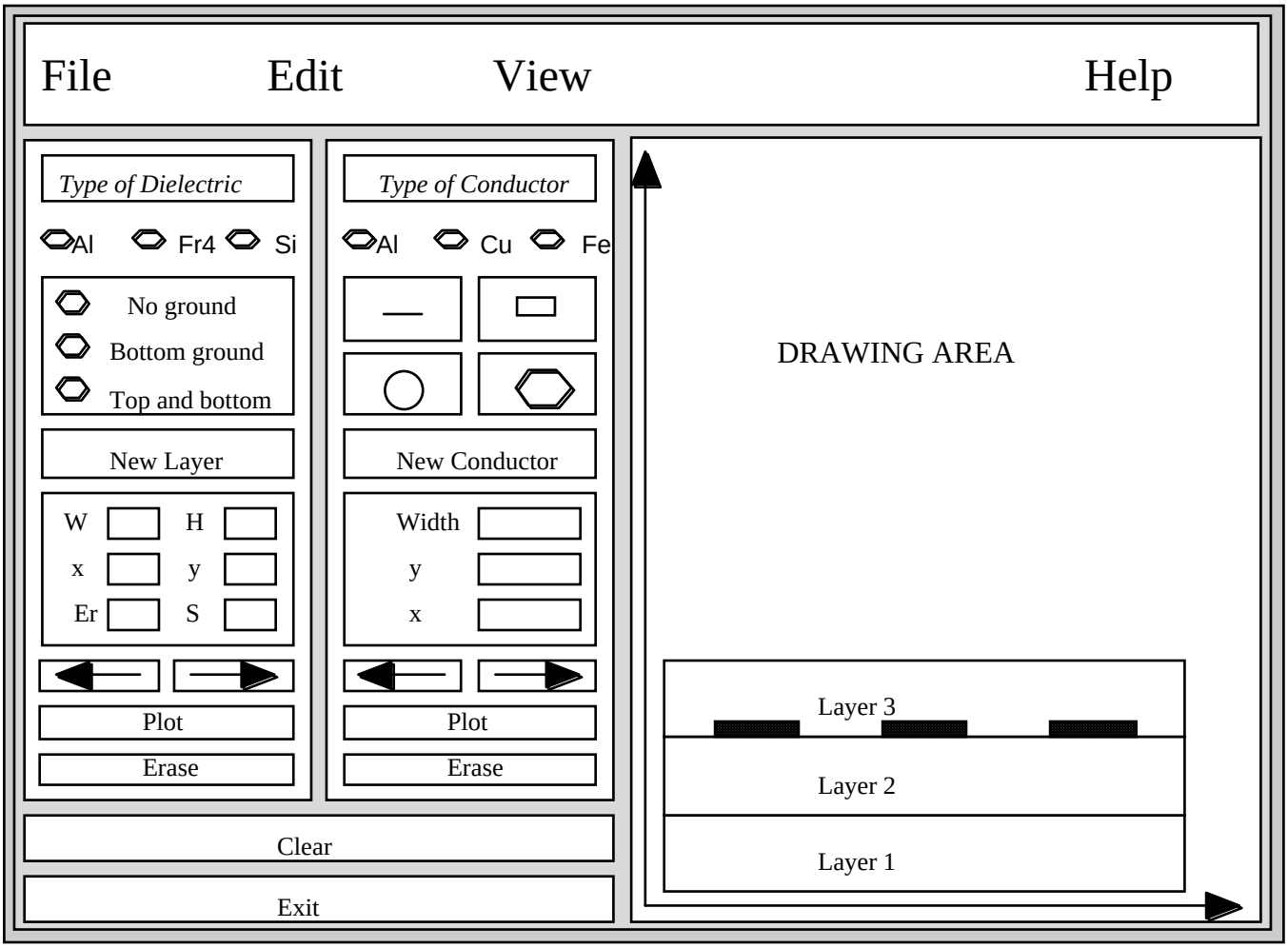

Fig 4.2: Main window

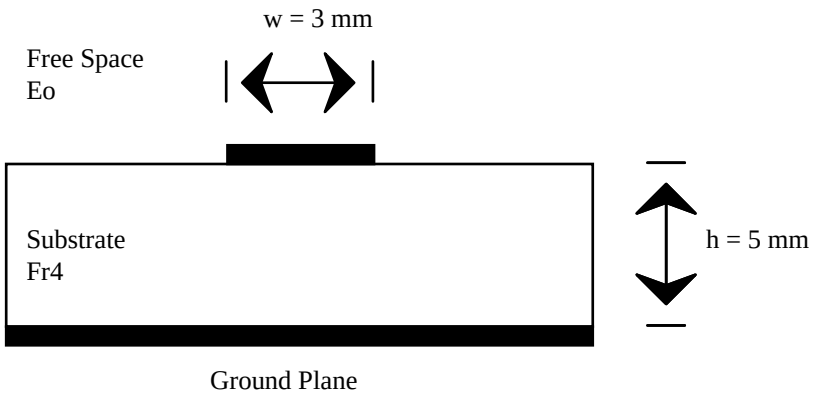

Fig. 4.3: A single microstrip line

The following description should be done for every structure considered:

Geometry and material :

one conductor:
strip, width: w = 3 mm, material: Fe
(x,y) = (15,5)
one dielectric layer:
height: h = 5 mm, material: Fr4 (permittivity $\varepsilon = 4.3\ \varepsilon o$, conductivity $\sigma = 0$)

This is a very good way to summarize the main characteristics of a given interconnect. The geometry and the material of each conductor and of each layer should be included when there is more than one conductor and more than one layer. The coordinates (*x*,*y*) of the lower-left corner of each conductor should be included as well. In the case of the strip, the value given to *x* is arbitrary because it is the first conductor. The value of *y* is deduced from the height of the first layer: $y = h = 5$ mm. Once this description is done, the design of the structure itself with the program XRLGC should be extremely simple.

4.2.2 Creating a new file

The option ‘New’ in the file menu allows the user to create a new file with the help of a custom dialog (Fig. 4.4). Once the dialog appears on the screen, the user may type the name of the file to create, for example, ‘Microstrip’ and then choose ‘OK’ to close the window. Another option would be to open an already existing file using the option ‘Open’ in the file menu. This will be seen in Section 4.5.

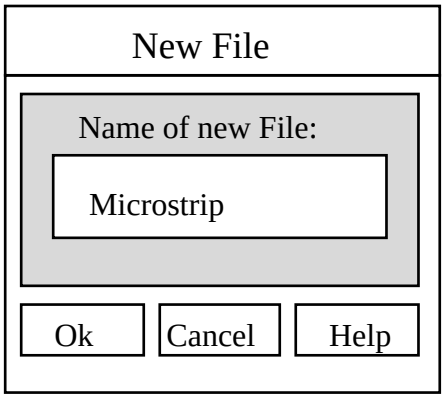

Fig. 4.4: New file window

4.2.3 Drawing the structure

In this example, the structure has only one bottom ground, so the option ‘bottom ground only’ on the layer column (Fig. 4.5), which is located on the left-hand side of the interface, should be chosen. Next, the unique layer will be created by selecting ‘New Layer.’ In order to get layers lying on top of each other, the (*x*,*y*) coordinates of the dielectric layer are automatically completed for each new layer created. Consequently, this first layer has coordinates (0,0). Second, the characteristics of the bottom layer (*h* = 5 mm, Fr4) should be completed on the form located towards the bottom of the layer column (Fig. 4.5). The user should select the option ‘Fr4’ among the various material located at the top of the layer column. The permittivity and conductivity of Fr4 ($\varepsilon_1 = 4.3\varepsilon_0$, $\sigma = 0$) will be automatically completed by the program on the form located at the bottom of the layer column. The user should then

Fig. 4.5

manually complete the dielectric height value in this same form. The button ‘Plot’ allows the user to see the new layer displayed on the drawing area.

The conductor column (Fig. 4.5) will be used next in order to design the strip itself. This part of the interface is located on the left of the drawing area and towards the center of the interface. First, the shape of the conductor is chosen among the four possibilities (strip, rectangle, circle, polygon) at the top of the conductor column. In our case, a microstrip line is a strip conductor placed on top of the dielectric; therefore we choose the shape ‘Strip.’ Once this is done, the new strip may be created by selecting ‘New Conductor.’ In order to draw a strip, some information about this conductor is needed: its position (x,y), its dimensions (width), and the material used. In Example 1, the width of the conductor is w = 3 mm, the position (x,y) = (15 mm, 5 mm), and the material is iron. Once this information is given, the user may click on ‘Plot’ to see the strip appear on the drawing area. Now that the structure is completely drawn on the screen, one can go to the next step: saving and starting the simulation.

4.2.4 Saving changes and creating the mesh files

First of all, the option ‘Save’ in the file menu should be selected in order to save the new file ‘Microstrip.’ Then, before starting the simulation, the user should select ‘Quality’ from the edit menu and indicate 50 basis functions (Fig. 4.6). The mesh file for the strip will then be created, according to the number of basis functions chosen, by selecting the option ‘Mesh Creation’ in the file menu. Shortly thereafter, the user will see a small window stating that the mesh has been created (Fig. 4.7). This working window may be closed by simply pressing ‘OK.’ Two new files have been created: ‘Microstrip’ and ‘strip1.mesh.’

4.2.5 Starting the simulation and looking at the results

The user may now choose the option ‘Solve’ from the file menu to simulate our single microstrip structure. Both *RLGC* parameters and potential distribution will be determined through this simulation. Once the RLGC simulation is finished, the potential simulation will begin. A new custom dialog will open, asking for the window coordinates used in the voltage simulation, the number of points in each direction, and the excited conductor number (see Fig. 4.8).

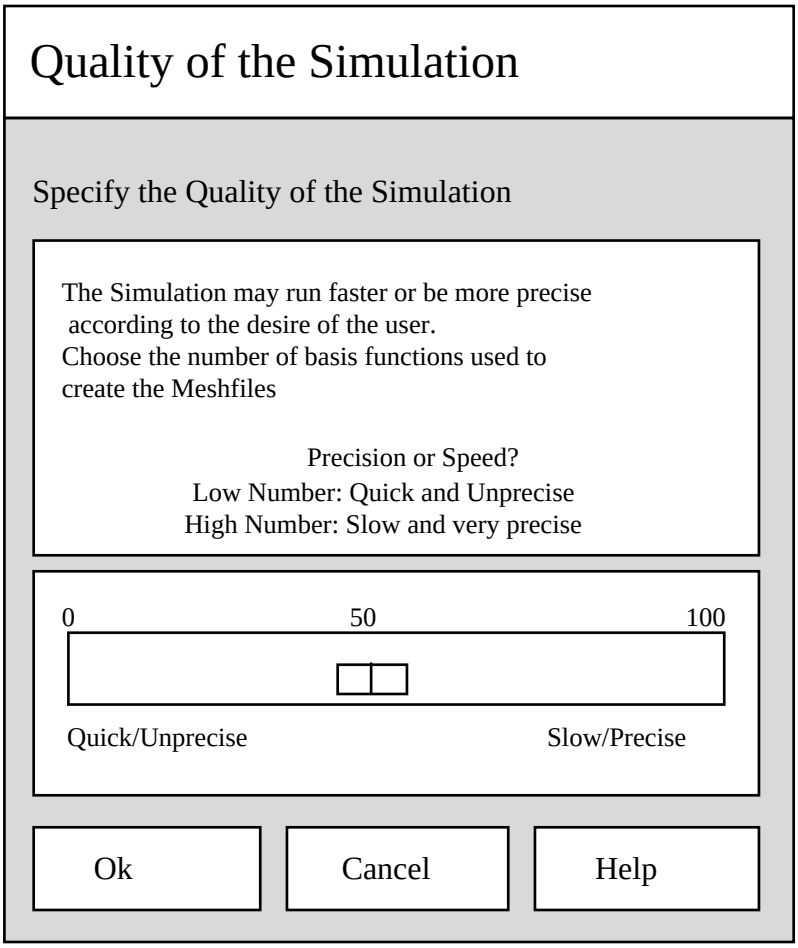

Fig. 4.6: Quality of the simulation

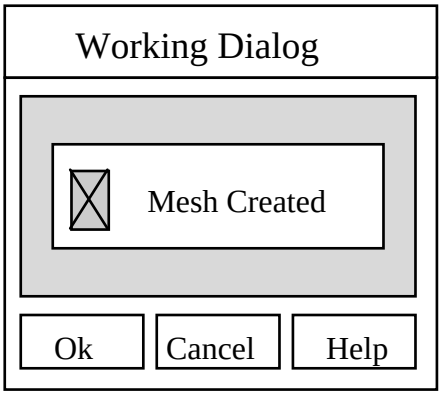

Fig. 4.7: Mesh creation window

If the user does not want to consider this second simulation but would prefere to look at the RLGC simulation results right away, it is possible to skip this part by pressing ‘Cancel’ instead of ‘OK’ in the voltage window. In the example given in Fig. 4.8, the bottom left corner of the simulation window is (0,0) and the top right corner is (10,10). The number of points in each direction is equal to 10. Furthermore, the microstrip is chosen to be the excited conductor. The conductor number corresponds to the order in which it has been created and displayed on the drawing area. The microstrip has been created first, and it is the only conductor present in the structure, so the conductor number of the microstrip is simply 1.

Once the necessary data is filled, the user shall click ‘OK.’ The simulation will then begin. Shortly thereafter, the user will see a small window stating that the simulation is done. The main results can be seen by choosing the option ‘RLGC’ in the view menu. A new window will then display the four matrices *R*, *L*, *G* and *C* (see Fig. 4.9). The potential distribution may be read in a separate file, ‘Microstrip.V2D.’ Refer to Section 5.1.3 for more information about the format of the ‘.V2D’ output files.

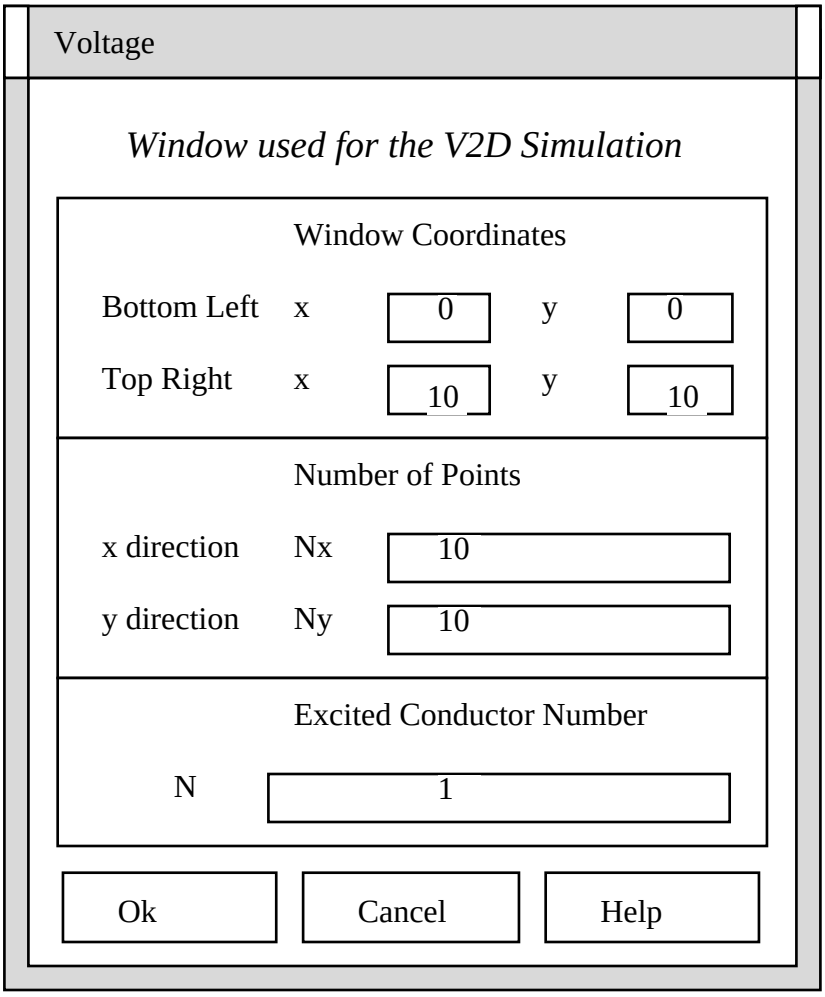

Fig. 4.8: V2D simulation dialog

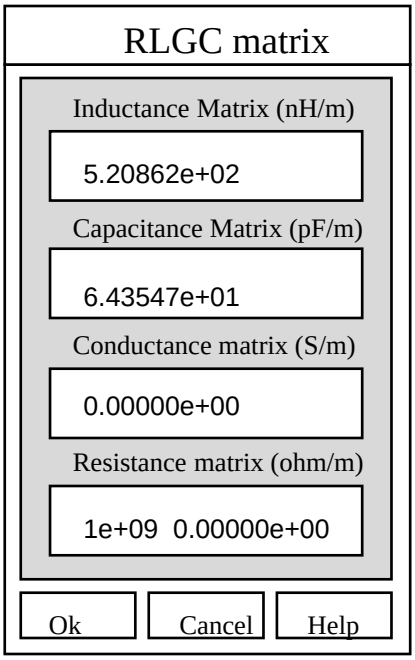

Fig. 4.9: RLGC custom dialog for the microstrip case.

4.3 Three Conductors in a Layered Medium

4.3.1 Hypothesis and geometry considered

Example 1 with one layer and one conductor showed the basic features of the GUI. This example will study the design of multilayered and multiconductor structures. We consider the structure shown in Fig. 4.10. We assume that the three rectangular conductors are identical, and the number of basis functions per side is chosen equal to 50 for each conductor. The coordinates (x_i ,y_i) of the three conductors can easily be deduced from Fig. 4.10, as will be seen in Section 4.3.3.

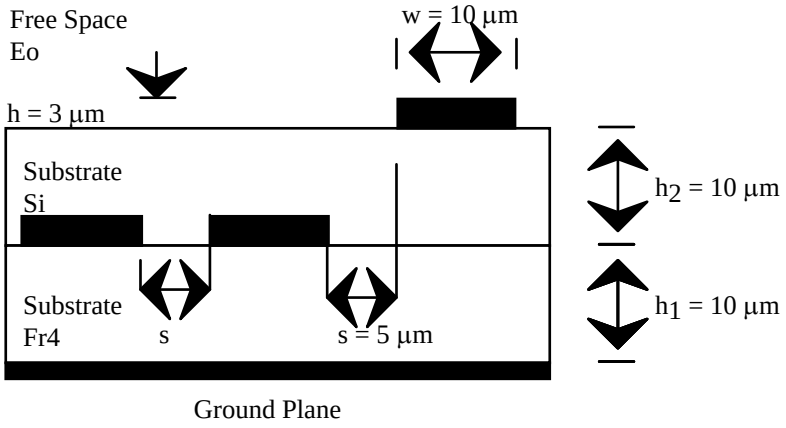

Fig. 4.10: Three rectangular conductors in a two-layer structure

The structure may be described as in Example 1 in Section 4.2.1:

Geometry and material:

three identical conductors:
width: w = 10 _m, height: h = 3 _m, spacing: s = 5 _m, and material: Cu
(x_1,y_1) = (0,10) _m
(x_2,y_2) = (15,10) _m
(x_3,y_3) = (30,20) _m
two dielectric layers
first layer: height h_1 = 10 _m, material Fr4 ($\varepsilon_1 = 4.3\varepsilon_0$, $\sigma = 0$)
second layer: height h_2 = 10 _m, material Si ($\varepsilon_2 = 3.9\varepsilon_0$ and $\sigma = 0.0016$)

4.3.2 Starting

The user may start as described in Section 4.2. However, if XRLGC is already loaded and a different circuit is displayed on the drawing area, three steps must be carried out. First, save this circuit by selecting ‘File-Save’. Then, clear the drawing area by clicking on the ‘clear’ button located towards the bottom of the interface. Finally, create a new circuit by selecting ‘File-New’ and naming the circuit—for our purposes, ‘ThreeRect.’

4.3.3 Designing the circuit

The first modification that needs to be done in this circuit is as follows. The user

changes the default length units to microns instead of millimeters. The user selects the option ‘Edit-Unit’ from the menu to open the corresponding dialog (see Fig. 4.11). Once the unit window is open, the user selects ‘Um’ for the length unit (‘Um’ stands for microns) and clicks OK. All the dimensions appearing in the GUI will then be microns by default.

In this example, the structure has only one bottom ground, so the option ‘bottom ground only’ should be chosen. Next, two layers will be created by using the layer column on the left-hand side of the interface. First, the user should select ‘New Layer’ to create the bottom layer. The user may note that the (*x*,*y*) coordinates of the dielectric layer are automatically completed for each new layer created, in order to get layers to lie on top of each other. Consequently, the first layer has coordinates (0,0), the second layer has coordinates ($0,h_1$), and the n^{th} layer ($0,\Sigma^{n}\_h_i$), where h_i is the height of the i^{th} layer. XRLGC cannot compute the parameters of interconnects involving a more complex dielectric structure. In order to have a successful simulation, the user should therefore not change the coordinate values (*x*,*y*) located at the bottom of the layer column.

Second, the characteristics of the bottom layer (h_1 = 10, Fr4) should be completed accordingly. The user should select the option ‘Fr4’ from the material located at the top of the layer column. The permittivity and the conductivity of Fr4 (ε_1 = 4.3ε_0 , σ_1 = 0) will be automatically completed by the program on the form located at the bottom of the layer column. The user should then manually complete the dielectric height value in this same form. The button ‘Plot’ allows the user to see the new layer displayed on the drawing area (Step 1 of Fig. 4.12).

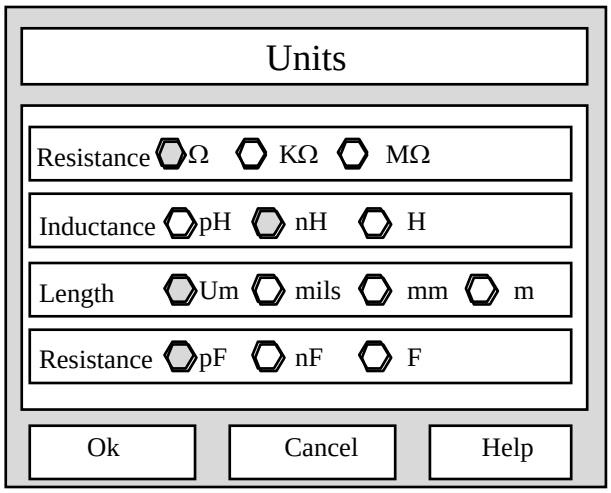

Fig. 4:11: Units dialog

The user follows similar directions to create the top layer (Step 2 of Fig. 4.12), with the parameters (h_2 = 10, Si). The other parameters—including position of the layer (x,y), permittivity, and conductivity—are filled automatically once the material is chosen. One layer is now positioned on top of the other.

Similarly, the conductor column is used to add the three rectangular conductors to the circuit. This process is easy and it only needs a short preliminary analysis of the structure considered; the user should be able to give the origin (lower left corner) and dimensions of each conductor. In this example, the width and height of each rectangular conductor are 10 μm and 3 _m. For the first conductor, x = 0 _m, y = h_1 = 10 _m; for the second conductor, x = w + s = 15 _m, y = h_1 = 10 _m; for the third conductor x = 2 (w + s) = 30 _m, y = h_1 + h_2 = 20 _m.

Once this is known, it is very easy to design the structure in the following way. The user chooses the rectangular shape (represented by a rectangle) on the top part of the conductor column and then clicks on ‘New Conductor.’

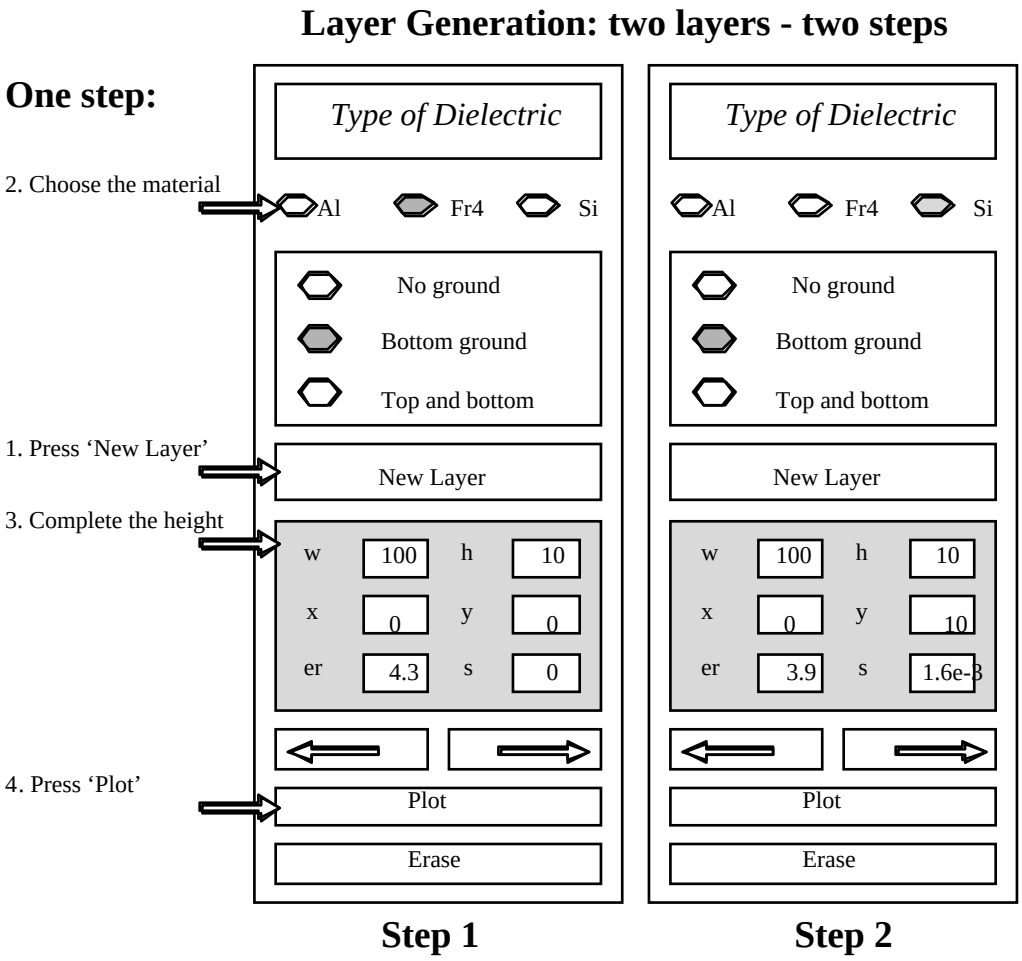

Fig. 4.12: Layer design

The material ‘Cu’ may then be chosen and the other parameters completed (x, y, width, height) according to the calculation done above for the first conductor. This conductor will then be displayed by clicking on ‘Plot’ towards the bottom of the conductor column. The user may repeat those steps for each rectangular conductor (see Fig. 4.13). In case an error is committed, it is possible to use the arrows to go back to the corresponding conductor. Once the error is corrected, the user may press ‘Plot’ again.

4.3.4 Saving and solving the circuit

The user may follow the guidelines given in Section 4.1: the file is saved using the option ‘File-Save’ from the menubar, the quality of simulation is chosen from the ‘Edit-Quality’ menu (50 for every conductor), and the mesh construction is started by selecting ‘Mesh Construction’ from the file menu. Once this is done, four files have been created: ‘ThreeRect,’ and the mesh files ‘Rect1.mesh,’ ‘Rect2.mesh,’ and ‘Rect3.mesh.’ These files do not need to be saved for future use.

The user may then simulate the structure by selecting the option ‘File-Solve’ on the menubar and completing the potential simulation dialog. Four additional files have been created in the process: ‘ThreeRect.in,’ ‘ThreeRect.out,’ ‘ThreeRect.Q2D,’ and ‘ThreeRect.V2D.’ The resulting matrices may be displayed by selecting the option ‘RLGC’ from the view menu. The *L*, *G*, and *C* matrices will be of dimension $3^{H}3$ because there were three conductors considered in the structure. If this structure was to be viewed or modified again in the future, only the file ‘ThreeRect,’ which contains the geometry of the interconnect, is required. However, the files ‘ThreeRect.out’ and ‘ThreeRect.V2D,’ which contain the *RLGC* matrices and the potential distribution, would be necessary in order to see the results of the last simulation.

4.4 Four Conductors in a Layered Medium

4.4.1 Hypothesis and geometry considered

We consider the structure shown in Fig. 4.14. We assume that the four rectangular conductors are identical and that the number of basis functions per side is 50 for each conductor.

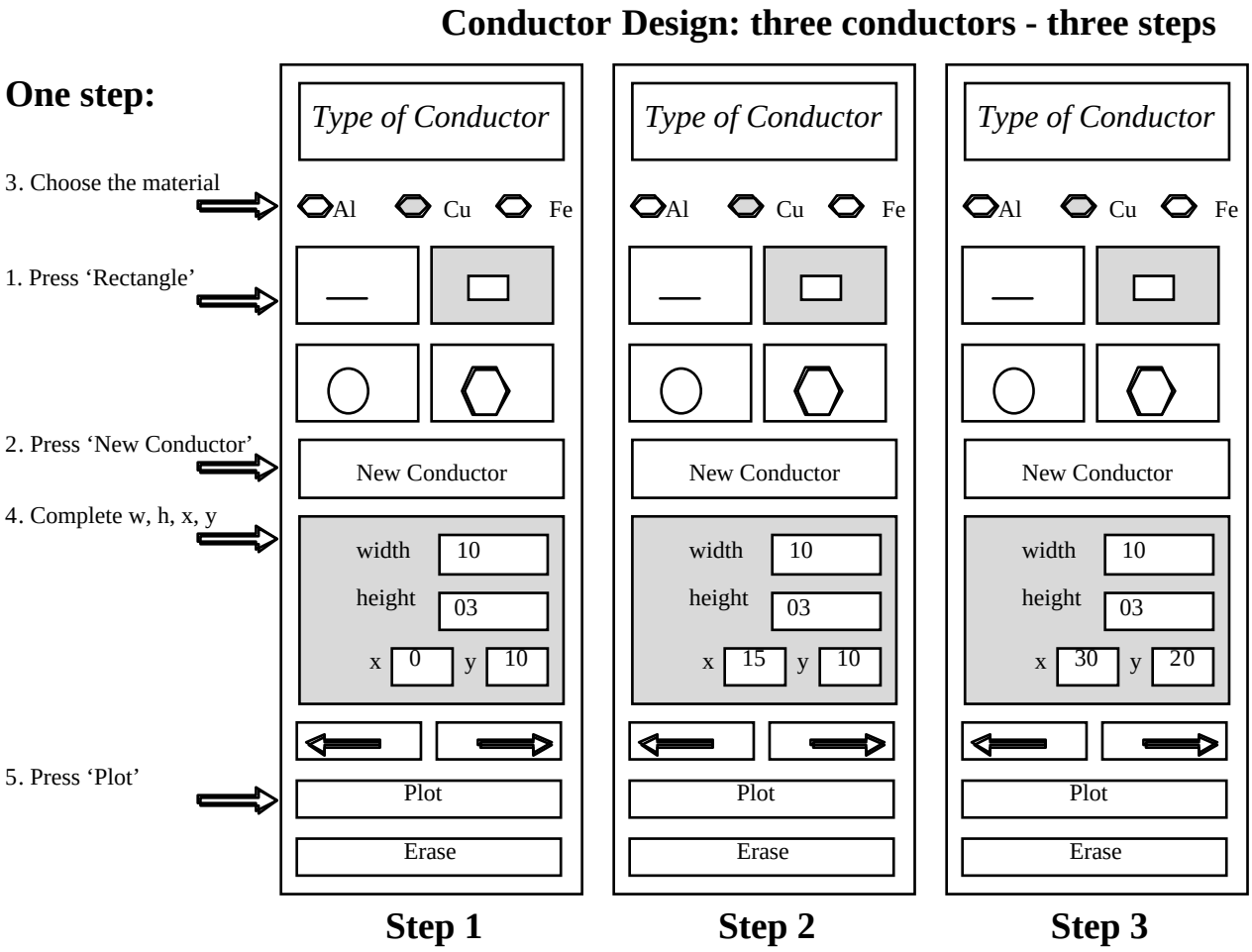

Fig. 4.13: Conductor design

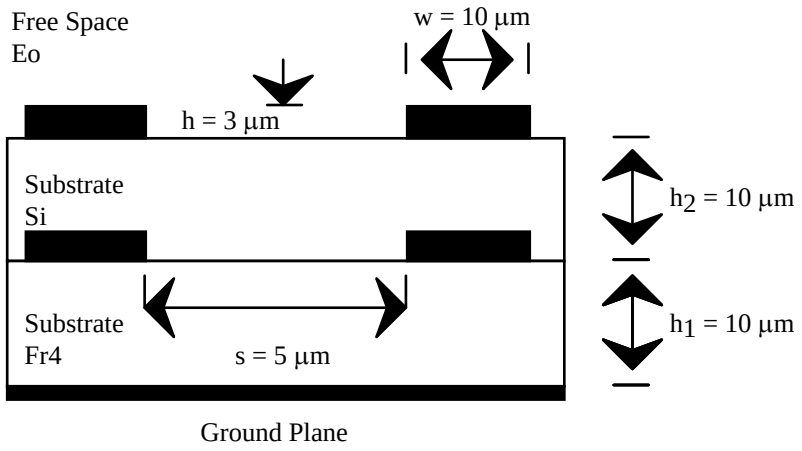

Fig. 4.14: Four rectangular conductors in a two-layer structure

The structure may again be described like Example 1 seen in Section 4.2.1:

Geometry and material:

four identical conductors:
width: w = 10 _m, height: h = 3 _m, spacing: s = 20 _m, and material: Cu
$(x_1,y_1) = (0,h_1) = (0,10)$ _m
$(x_2,y_2) = (w + s,h_1) = (30,10)$ _m
$(x_3,y_3) = (w + s,h_1 + h_2) = (30,20)$ _m
$(x_4,y_4) = (0,h_1 + h_2) = (0,20)$ _m
two dielectric layers
first layer: height h_1 = 10 _m, material Fr4 ($\varepsilon_1 = 4.3\varepsilon_0$, $\sigma = 0$)
second layer: height h_2 = 10 _m, material Si ($\varepsilon_2 = 3.9\varepsilon_0$ and $\sigma = 0.0016$)

4.4.2 Starting

It would be possible to design and simulate this third example in a way similar to the two preceding examples. This would imply creating from scratch a new file and designing the two layers and the four conductors. However, a better solution will be exposed in this section.

The structure considered is similar to the three-conductor example solved in Section 4.3. First, the layered structure is absolutely identical. Second, two of the conductors are the same and the two others have the same dimensions but different coordinates. Therefore, this interconnect may be easily designed by slightly modifying the file ‘ThreeRect’ created in Section 4.3. The design of the structure may therefore be started by opening the file

‘ThreeRect.’ The window shown in Fig. 4.15 will appear on the screen once the option ‘Open’ in the file menu has been chosen. The user may select the file ‘ThreeRect’ and then close the window with ‘OK.’

The next step involves opening a new file under the name of ‘FourRect.’ The user may select the option ‘New’ from the file menu, type in *FourRect* and select ‘OK’ to close the window. The drawing area still contains the structure of the file ‘threerect’ with the two layers and the three conductors. In order to save this structure under the name ‘FourRect,’ the user may then select the option ‘Save’ from the file menu.

4.4.3 Designing the Circuit

For now, the file ‘FourRect’ contains the two layers and the three conductors that were involved in the structure of ‘ThreeRect.’ The goal of this section is to design the four-conductors structure given in Fig. 4.14, starting from this three-conductors structure. Obviously, the default length units should be changed to microns instead of millimiters (see Section 4.3.3). The geometry of the dielectric layers has not changed, so no modification is needed. However, the geometry of the conductors is similar but not identical. Conductors 1 and 3 are identical (see Fig. 4.16), but Conductor 2 has different coordinates: (25,10) instead of (10,10). Moreover, Conductor 4 needs to be completely designed.

The user will therefore check each conductor successively with the help of the arrows located at the bottom of the conductor column, starting with Conductor 1. The geometry of Conductor 1 (w = 10 _m, h = 3 _m, x = 0, y = 10 _m) is adequate, the length unit being now set to microns. Consequently, the conductor 1 is not modified, and the right arrow may be used to look at the second conductor.

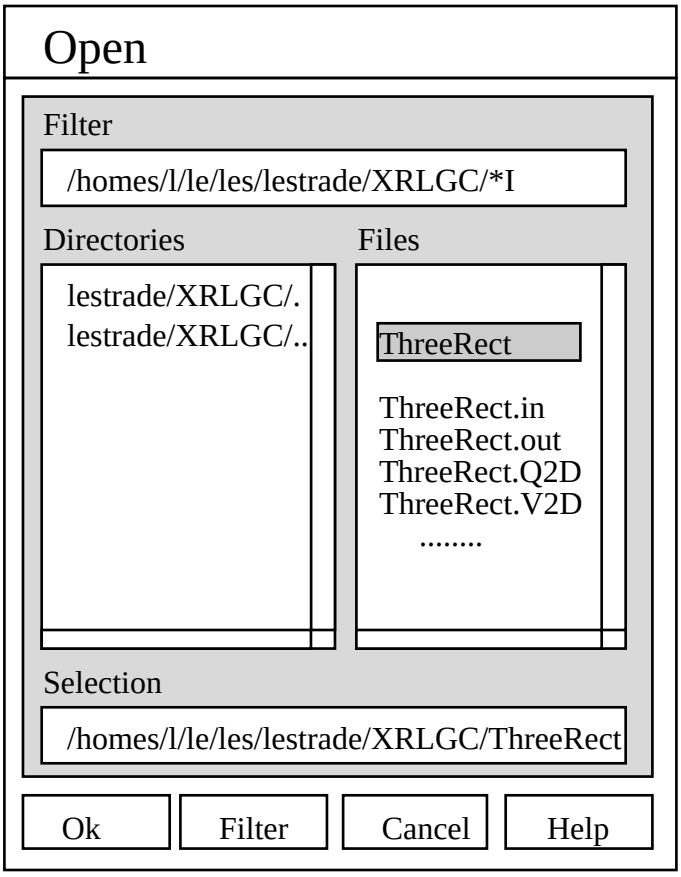

Fig. 4.15: Open file window

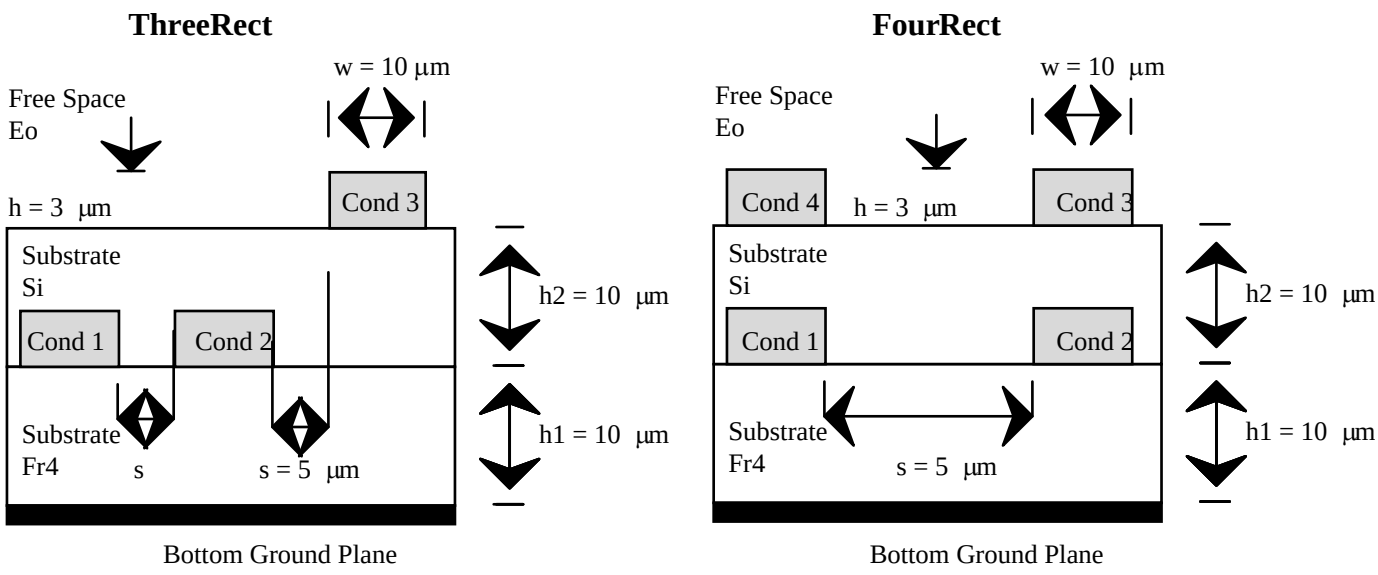

Fig. 4.16: Comparing the structures of ThreeRect and FourRect

The second conductor has the right width and height (w = 10 _m, h = 3 _m), but the coordinates (x,y) need to be changed from (15,10) to (30,10). Once those new coordinates are typed in and the button 'Plot' located on the conductor column is pressed, the modified second conductor will appear on the screen. The right arrow may be used again to check the correctness of Conductor 3. The geometry of Conductor 3 (w = 10 _m, h = 3 _m, x = 30 _m, y = 20 _m) being adequate, the next step is to create the fourth conductor. In order to design this last conductor, the user selects 'New Conductor,' types in the four parameters (w = 10 _m, h = 3 _m, x = 0, y = 20 _m), and presses 'Plot.'

4.4.4 Saving and solving the circuit

The four conductors are now displayed on the drawing area according to the structure shown in Fig. 4.14. This structure may be saved by selecting the option 'Save' from the file menu. The mesh file construction and simulation are done quite similarly to Section 4.3.2, and they result in the creation of nine files: 'FourRect,' 'FourRect.in,' 'FourRect.out,' 'FourRect.Q2D,' 'FourRect.V2D,' and the mesh files 'Rect1.mesh,' 'Rect2.mesh,' 'Rect3.mesh,' and 'Rect4.mesh.'

The *R*, *L*, *G*, and *C* matrices are obtained by selecting the option 'RLGC' in the view menu. The *L*, *G* and *C* matrices will be of dimension $4^{H}4$ because there were four conductors considered in the structure. If this structure was to be viewed or modified again in the future, only the file 'FourRect' (containing the geometry of the interconnect) and optionally the files 'FourRect.out' and 'FourRect.V2D' (containing the results of the last simulation) would be required.

4.5 Conclusion

Various structures have been discussed in this chapter, involving different types of conductors and dielectrics. The first example, including one conductor and one layer only, gave a quick insight on the basic use of XRLGC. The second example extended the use of XRLGC to a more complex structure involving multiple layers and multiple conductors. Finally, the third example showed how an already existing structure may be modified to easily obtain another structure. These three examples implied the use of strips and rectangular conductors, but the use of circular conductors would be identical. It is only essential to know that the shape of the conductor should be chosen before the new conductor in that particular shape is created. Moreover, the arrows may only be used to go back and forth among

conductors of the same shape, strip, rectangle, or circle. If, for example, a conductor of circular shape needs to be modified, the circular shape should be selected first. Then the arrows may be used accordingly. Similarly, a layer may be modified by using the arrows located in the layer column. Unfortunately, if the user modifies the height of a layer, the layer coordinates (x,y) are no longer completed automatically in order to get layers lying on top of each other. Consequently, those coordinates have to be completed by the user.

CHAPTER 5
STRUCTURE OF XRLGC

This section will discuss the original program RLGC written by K. S. Oh before adding the interesting elements inherent to XRLGC and its GUI. The RLGC package is made out of three main programs. First, Mesh2D creates a mesh file for one conductor, given its geometry and the number of basis functions considered. This program is called for each conductor of the structure and the output is a '.mesh' file. The second program, RLGC, computes the *RLGC* matrix and the charge distribution for the multiconductor, multilayer structure described in an input file ('.in'). Those results are stored in two separate files '.out' and '.Q2D.' The input file is manually completed by the user. The third program, V2D, computes the potential distribution given the same input file ('.in') and the charge distribution ('.Q2D').

The program XRLGC presented in this thesis links the three programs using a GUI. The circuit is displayed on a drawing-area and can be easily modified by the user. The processing of the transmission line parameters is done in two steps: mesh generation for each conductor (menu: 'File-Mesh') and solving (menu: 'File-Solving'). Solving involves, first, automatic creation of the input file containing all the information about layers and conductors that have been collected by the interface and, second, computing the *RLGC* matrix, charge distribution, and potential distribution.

With the file menu, the user can also create a new file, open an already existing file, and save the changes made to the circuit ('File-New'/'Open'/'Save' options). The file contains the geometry of the multilayered, multiconductor structure created. With the edit menu, the user can choose the different units (distance, inductance, capacitance, resistance), determine the quality of simulation (number of basis functions used), and choose some parameters used for potential computation.

The *RLGC* matrix obtained through the simulation may also be displayed using the option 'View-RLGC.' The potential and charge distributions are obtained in separate files that will be eventually displayed by the GUI. With the help menu, the user has an index and contextual help available.

5.1 Structure of RLGC

5.1.1 Overview

The files included in the source of RLGC are following:

C-source: RLGC.c, Mesh2D.c, V2D.c, appx.c, intg.c, lu.c, aux.c

H-files: RLGC.h, aux.h, ansi_compat.h

The original makefile (see Fig. 5.1) compiles separately the three different programs: Mesh2D, RLGC, and V2D. These three programs are then used consecutively to get the RLGC parameters and the charge and voltage distributions.

```
# Original makefile (RLGC)
# You need to change this path
BIN_DIR = /mnt/users_decwd/ksoh/bin
default:
        cc -g -o RLGC RLGC.c lu.c appx.c intg.c aux.c -lm
        mv RLGC $(BIN_DIR)
        cc -o Mesh2D Mesh2D.c aux.o -lm
        mv Mesh2D $(BIN_DIR)
        cc -o V2D V2D.c appx.o aux.o intg.o -lm
        mv V2D $(BIN_DIR)
clean:
        rm *.o
```

Fig. 5.1: Original makefile for the RLGC package.

5.1.2 Constructing the mesh files and input file of RLGC

In order to construct the mesh files, the Mesh2D program is called once for each conductor of different shape and size. A sample running of Mesh2D and the resulting mesh file are given in Figs. 5.2 and 5.3. The program asks for the type of conductor desired, the name of the file to create, the dimensions of the conductor, and the precision of the simulation measured by the number of basis functions. In the example given here, the conductor is rectangular, of width 5 mm, height 5 mm, and only 5 basis functions on each side (poor precision). The mesh file of Fig. 5.3 contains the coordinates of points evenly spaced on the four borders of the rectangular conductor. The points are considered two by two, starting from (0,0) (0,1),

```
> Mesh Construction
       1) Strip
       2) Rectangular
       3) Polygon
       4) Circle
       5) quit

> Enter the Selection: 2
> Enter the mesh file name to be created: rect.mesh
> Enter width and height of line in [mm]: 5 5
```

```
> Enter # of basis functions for width and height: 5 5
```

Fig. 5.2: Sample running of Mesh2D

moving along each border, and turning clockwise around the rectangular conductor. There are five points per side according to the number of basis functions chosen. The structure of mesh files for strips or circular conductors is similar.

This program needs to be called a few times to get all the mesh files for the structure considered. Once the mesh files for each conductor are created, the next step is to manually fill out the input file of the second program, RLGC. This input file contains the overall structure of the multiconductor multilayered transmission line. The user carefully completes the number and dimensions of the conductors and layers, according to certain standards defined at the beginning of the template file (see Fig. 5.4).

The example given in Fig. 5.4 considers one circular conductor of resistivity 1.73e-8 Ω^{H}mm, radius 0.5 mm, located at (x,y) = (0,3 mm), and included in a single layer of relative permittivity 4 and conductivity 2.67e-5 S/m. The structure has only one bottom ground.

5.1.3 Running the simulation

Once the input file ‘circuit.in’ is completed and all the mesh files are constructed, the simulation is executed by running RLGC and V2D successively. First, RLGC is started by typing

> RLGC circuit.in.

The program RLGC computes the four parameters resistance, inductance, conductance, and capacitance of the structure included in the input file (Fig. 5.4). Those parameters are given in the output file ‘circuit.out’ (Fig. 5.5).

```
Rect.mesh:
UnitFactor 1000
NumberOfBasisFns 20
MeshType Rectangular 5.00000e+00 5.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 1.00000e+00
0.00000e+00 1.00000e+00 0.00000e+00 2.00000e+00
0.00000e+00 2.00000e+00 0.00000e+00 3.00000e+00
0.00000e+00 3.00000e+00 0.00000e+00 4.00000e+00
0.00000e+00 4.00000e+00 0.00000e+00 5.00000e+00
0.00000e+00 5.00000e+00 1.00000e+00 5.00000e+00
1.00000e+00 5.00000e+00 2.00000e+00 5.00000e+00
2.00000e+00 5.00000e+00 3.00000e+00 5.00000e+00
3.00000e+00 5.00000e+00 4.00000e+00 5.00000e+00
4.00000e+00 5.00000e+00 5.00000e+00 5.00000e+00
```

```
5.00000e+00 5.00000e+00 5.00000e+00 4.00000e+00
5.00000e+00 4.00000e+00 5.00000e+00 3.00000e+00
5.00000e+00 3.00000e+00 5.00000e+00 2.00000e+00
5.00000e+00 2.00000e+00 5.00000e+00 1.00000e+00
5.00000e+00 1.00000e+00 5.00000e+00 0.00000e+00
5.00000e+00 0.00000e+00 4.00000e+00 0.00000e+00
4.00000e+00 0.00000e+00 3.00000e+00 0.00000e+00
3.00000e+00 0.00000e+00 2.00000e+00 0.00000e+00
2.00000e+00 0.00000e+00 1.00000e+00 0.00000e+00
1.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
```

Fig. 5.3 : Sample output file of Mesh2D (‘rect.mesh’)

RLGC also determines the charge distribution included in the ‘circuit.Q2D’ file. The charge distribution is given in the following format: for each pair of points (x_1,y_1), (x_2,y_2) defined in the mesh file, the value of the charge q_{12} is determined. This result is used by the next program V2D to compute the potential distribution. The program V2D is started by typing

> V2D circuit.in.

V2D will ask for certain parameters as well: size of the window considered, number of points in each direction, and index of the excited conductor. A simple running of V2D is given in Fig. 5.6 The resulting file ‘circuit.V2D’ contains the potential distribution according to the format given in Fig. 5.7.

```
*----------------- Template for the main input file ----------------------
*                                                                  Kyung S. Oh
*                                                                  3/24/94
*-------------------------------------------------------------------------
*  Convention Used:
*     o  x y coordinates are used in the program.
*     o  If there is ground plane at the bottom, it is at y=0.
*     o  The conductor numbers used in the program are ordered as listed
*        in this file.
*     o  The dielectric number is assigned from bottom to top.
*     o  Conductivity Unit S/m
*     o  Resistivity Unit ohm*mm
*     o  All parameters associated with length are in mm.
*-------------------------------------------------------------------------
*  Number_of_conductors    N
*      mesh_file_name1     Resistivity x  y
*      mesh_file_name2     Resistivity x  y
*      mesh_file_nameN     Resistivity x  y
*  Note: The mesh file names do not have to be distinct.
*  x,y is the global coordinate of the local origin for each mesh file.
Nconductors        1
circle.mesh       1.73611111e-8 0.0 3.0
*-------------------------------------------------------------------------
*  Number of layers without ground planes
Nlayers           1
*-------------------------------------------------------------------------
*  Ground plane options:
*       0 for no ground plane
*       1 at the bottom layer only
*       2 for ground planes  at top and bottom
*  GND    GND_option  Resistivities(iff any)
*  (ex)   GND  0        GND 1 Rvalue    GND 2 Rvalue_lower Rvalue_upper
GND               1  0
*-------------------------------------------------------------------------
*  The dielectric constants of the layers
Er                4.0
*-------------------------------------------------------------------------
*  The conductivity of the layers
Conductivity      2.67e-5
*-------------------------------------------------------------------------
*  ds are the distances from the ground plane to dielectric interfaces.
*    The bottom ground plane is assumed to be at y=0.
*    The value of the last entry is the location of the top ground plane.
*    If there is no top ground plane, enter an arbitrary number.
ds(mm)             1e10
*-------------------------------------------------------------------------
* frequency : fstart fstop fstep
Frequency 0.1e9 0.1e9 0.1e9
*-------------------------------------------------------------------------
```

Fig. 5.4: Template for the input file of RLGC (‘circuit.in’)

```
#circuit.out
Capacitance Matrix (pF/m)
 8.97505e+01
Inductance Matrix (nH/m)
 4.95211e+02
Conductance Matrix (S/m)
 6.76611e-05
Resistance Matrix (ohm/m)
1e+08 8.44559e-04
```

Fig. 5.5: Sample of an output file of RLGC (‘circuit.out’)

```
>V2D circuit.in
> Enter X,Y coord of the BOTTOM LEFT corner of window(mm): 0 0
> Enter X,Y coord of the TOP RIGHT corner of window(mm): 10 10
> Enter number of point in x and y direction: 10 10
> Enter the excitated conductor number: 1
```

Fig. 5.6: Sample running of V2D

```
0 Nx Ny Dx Dy Xstart Ystart Xstop Ystop
V(x(1),y(1)) V(x(2),y(1)) .... V(x(Nx),y(1))
V(x(1),y(2)) V(x(2),y(2)) .... V(x(Nx),y(2))
...
V(x(1),y(Ny)) V(x(2),y(Ny)) .... V(x(Nx),y(Ny))
```

Fig. 5.7: Format of an output file of V2D (‘circuit.V2D’)

5.1.4 Technical file description

Now that we have seen how the three programs work and how they interact to compute the final results, it is interesting to look at the corresponding files one by one. ‘RLGC.c,’ ‘Mesh2D.c,’ and ‘V2D.c’ are, respectively, the main routines for the three programs described above. The four additional C-files contain a set of tools useful to those main programs. ‘Appx.c’ generates the closed-form Green’s function. ‘Intg.c’ is a set of integration routines. ‘Lu.c’ includes the LU factorization and forward and backward routines. ‘Aux.c’ contains the utility functions to create the vectors and matrices. The Mesh2D program basically uses ‘Mesh2D.c’ and ‘aux.c’; the V2D program uses ‘V2D.c,’ ‘aux.c,’ ‘appx.c,’ and ‘intg.c’; and the RLGC program uses ‘RLGC.c’ and all four additional C-files: ‘aux.c,’ ‘appx.c,’ ‘intg.c,’ and ‘lu.c.’ The resource file ‘aux.h,’ defining some structures and declaring the functions included in ‘aux.c,’ is used in the three programs Mesh2D, RLGC, and V2D.

The resource file ‘RLGC.h’—declaring the functions included in ‘RLGC.c,’ ‘V2D.c,’ ‘appx.c,’ ‘lu.c,’ and ‘intg.c’—is only used in RLGC and V2D.

The package RLGC including these three main programs was therefore rather difficult to use. The user often needed to run them more than once before getting any satisfying results. If the user wished to make even minor changes in the shape or size of the conductors and layers, all three programs had to be run all over again, wasting time. The user also needed to generate the mesh for each conductor, sometimes by running Mesh2D many times. Moreover, the use of RLGC implied manually changing the input file, changing units adequately before filling in the geometry of the structure. The slightest error in this manual input would automatically lead to wrong results at the output.

5.2 XRLGC New Features

5.2.1 Overview

The files included in the source of XRLGC are the following:

C-source: RLGC1.c, Mesh2D.c, V2D.c, appx.c, intg.c, lu.c, aux.c
frontend14.c, graphtool6.c, manageform.c, cond_call.c,
layer_call.c, apply.c, cust_dialog3.c, help_dialog.c, toolbar.c

H-files: RLGC.h, aux.h, ansi_compat.h,
apply.h, cond_call.h, layer_call.h, math.h,
graphtool.h, manageform.h, mytool.h

The makefile (see Fig. 5.8) compiles one unique program, XRLGC, which will compute the *RLGC* parameters and the charge and voltage distributions, according to the detailed description of the multilayered and multiconductor structure that has been collected by the interface.

```
# Makefile (XRLGC)

SOURCES=frontend14.c   mytool.c   graphtool6.c   manageform.c   cond_call.c
layer_call.c  RLGC1.c  lu.c  appx.c  intg.c  aux.c  apply.c  cust_dialog3.c
```

```
help_dialog.c toolbar.c V2D.c
OBJECTS=frontend14.o   mytool.o   graphtool6.o   manageform.o   cond_call.o
layer_call.o  RLGC1.o  lu.o  appx.o  intg.o  aux.o  apply.o  cust_dialog3.o
help_dialog.o toolbar.o V2D.o
CC=gcc
CFLAGS=-g -I.
LFLAGS=-lXm -lXt -lX11 -lm
PRODUCT=XRLGC
all:result
result: $(OBJECTS)
        $(CC) $(CFLAGS) -o $(PRODUCT) $(OBJECTS) $(LFLAGS)
clean: rm *.o
```

Fig. 5.8 : Makefile for XRLGC

5.2.2 Technical file description

Sources of XRLGC include files that were part of the original RLGC package. Those files have been slightly modified in order to communicate with the interface during the simulation and to give some warning to the user if some data are missing and the simulation could not be fulfilled. Besides those small changes, the files ‘RLGC.c,’ ‘V2D.c,’ ‘Mesh2D.c,’ ‘aux.c,’ ‘appx.c,’ ‘lu.c,’ and ‘intg.c’ are basically the same as in the original package.

The description of each C-file and H-file that is linked to the interface itself follows. The variables and functions used in XRLGC are all defined in the file ‘graphtool.h.’ The file ‘frontend14.c’ contains the resources (colors, fonts, label) of all the C-Motif widgets and the main function called by XRLGC. The main function defines the interface and most of the widgets associated with it. However, ‘graphtool6.c’ helps to unload the main function of ‘frontend14.c’ by taking care of the creation of certain complex forms and their children widget—for example, the form containing the drawing area or the form displaying the four different shapes of conductors (strip, rectangular, circular, polygonal).

The functions included in ‘manageform.c’ manage the adequate form depending on the type of conductor chosen (strip, rectangular or circular). This form asks for the dimensions (width, height, radius) and position (x,y) of the conductor. In the case of the strip, the width and position (x,y) are sufficient to model the conductor. Similarly, the rectangular conductor is characterized by its width, height, and coordinates (x,y), while a circular conductor is characterized by its radius and coordinates (x,y).

The file ‘cond_call.c’ is only concerned with the conductors—creating and erasing a conductor, and going back and forth between conductors to modify them. Similarly, the file

‘layer_call.c’ deals only with the layers—creating and erasing a layer, and going back and forth between the layers to modify them.

The functions included in ‘apply.c’ run all the different parts of the simulation: mesh construction (formerly Mesh2D), computation of the *R*, *L*, *G*, and *C* parameters (formerly RLGC), and estimation of the potential distribution (formerly V2D). This file constitutes the main link between the GUI and the original RLGC.

The file ‘cust_dialog3.c’ creates all the custom dialogs, except the help dialog: introduction dialog, units dialog, quality dialog, RLGC dialog, and voltage dialog. These complex dialogs have been designed specifically for this program. The more simple dialogs like the working dialogs or the prompt dialogs are already partially implemented by Motif. The file ‘help_dialog.c’ creates the help dialog and associated functions. Finally, ‘toolbar.c’ put adequate bitmaps on pushbuttons, including the strip, rectangle, circle, and polygon, representing the possible cross-sections for a conductor. This file also inserts the arrows to go back and forth between conductors and layers.

A more detailed description of those different files and the functions included is given inside the sources themselves. The goal of each function is given in the form of commentaries embedded inside the C-files.

5.2.3 File interaction

In the preceding section, each file has been described in detail. In this section, the interaction between those files will be clarified through a simple example. The first example seen in Section 4.2 will be considered to explore how each widget included in the interface does its task. Fig. 5.9 includes all the action performed by the user. In the layer column, the option ‘bottom ground only’ has first been selected, calling the function ‘toggleCB’ in the file ‘frontend.c’ and basically updating the variable ‘ground’ to the Value 1. The Value 0 means no

Fig. 5.9

ground, the Value 1 means a bottom ground, and finally the Value 2 means both bottom and top ground. This value is automatically recorded by the interface.

Then, a new layer has been created by clicking on ‘New Layer,’ consequently calling the function ‘New_Layer_callback’ in the file ‘layer_call.c.’ This function updates the counter ‘compteur_layer’ and the total number of layers ‘nombre_layer’ while it completes the form located at the bottom of the layer column in order to get layers located on top of eachother. The counter is useful when the user decides to move from one layer to the other with the arrows ‘—>’ and ‘<—.’ The counter tells the interface at which index it should look inside the layer arrays to find the right layer (see below).

Thereafter, the type of layer ‘Fr4’ may be selected at the top of the layer column. The function ‘toggleCB’ in ‘frontend.c’ is called again, setting the variable ‘current_type_layer’ to 3 and completing the form located at the bottom of the layer column to get a relative permittivity of 4.3. The user may then complete the height of the new layer (h = 5 m) and click on ‘Plot.’ This push-button will then call the callback function ‘store_data_layer’ in ‘frontend.c.’ This function is a rather complex procedure that involves updating many variables and the display of the new layer on the drawing area. The function ‘store_data_layer’ updates the layer arrays ‘Width_layer[],’ ‘Height_layer[],’ ‘x_layer[],’ ‘y_layer[],’ ‘e_layer[],’ and ‘s_layer[]’ accordingly at the index ‘compteur_layer.’ In case the button ‘New_layer’ had not been called, the layer located at this index simply would have been erased and replaced by this new layer with its new characteristics. In order to display the new set of layers, the full window is cleared and everything is drawn again. In this example, this set of actions led to the display of the layer in the drawing area.

Now, the microstrip is added to the structure in the following way. The selection of the strip shape calls the function ‘manage_strip’ in the file ‘manageform.c.’ This function manages the correct form ‘form_data_strip’ located at the bottom of the layer column, so that it asks for the coordinate (x,y) and width of the strip only. The three forms ‘form_data_strip,’ ‘form_data_rect,’ and ‘form_data_circle’ are already created when XRLGC is started, and they are managed or unmanaged as needed.

This form asks for the dimensions (width, height, radius) and position (x,y) of the conductor. In case of the strip, the width and position (x,y) are sufficient to model the

conductor. Similarly, the rectangular conductor is characterized by its width, height, and coordinates (x,y), while a circular conductor is characterized by its radius and coordinates (x,y).

The new strip is then created by selecting ‘New Conductor.’ This push-button calls the function ‘New_Cond_callback’ in the file ‘cond_call.c.’ This function updates the conductor counter ‘compteur’ and the total number of conductors ‘nombre_conducteur’ while it completes the form located at the bottom of the conductor column with zeros. The counter tells the interface at which index it should look inside the conductors arrays to find the right conductor (see below). The selection of the material ‘Fe’ calls again the function ‘toggleCB’ in frontend.c and updates the variable ‘current_material_cond’ to the value 3. Each material is recorded with a different value so that the interface can record which material has been chosen for that specific conductor.

Once the values (x,y) and width are completed, the user may select ‘Plot’ to display the conductor. This push-button will then call the callback function ‘store_data’ in ‘frontend.c,’ which is very similar to ‘store_data_layer’ except that it is dealing with conductors instead of layers and that there are three types of conductors to consider: strip, rectangular, and circular. The function ‘store_data’ updates the arrays ‘Width[],’ ‘Height[],’ ‘x[],’ ‘y[],’ ‘resistivity[],’ and the ‘indice[]’ accordingly at the index ‘compteur.’ The array ‘indice[]’ indicates what kind of conductor is considered—strip, rectangular, or circular. The color of the displayed conductor is directly linked to the material and is stored in the array ‘FillColor[]’ at the index ‘compteur.’ This function also updates the three counters ‘compteur_s,’ ‘compteur_r,’ and ‘compteur_c’ for each kind of conductor. These counters are essential when the user wants to go back and forth between conductors of the same shape. The microstrip should be now displayed on the drawing area as well as the layer.

The completed structure may now be saved through the file menu. The user will first select the option ‘New’ in the file menu, leading to the opening of a simple prompt dialog asking for the name of the new file. The function ‘NewFile’ in ‘frontend.c’ records the name of the new file ‘circuit.’ Second, the selection of the option ‘Save’ in the file menu will call the function ‘Save_File’ in frontend.c and thus create the file named ‘circuit’ according to precise rules given in Fig. 5.10. The full geometry of the structure will be saved in this file, so that the structure may be opened again in the future: by selecting the option ‘Open’ in the file menu, the user is calling the functions ‘OpenFile’ and indirectly ‘read_file’ in ‘frontend.c.’ This last function will read the file created above and display the corresponding structure. It

will also update all the arrays for the conductors and the layers that have been defined earlier.

```
Ground options:
GND 0,1 or 2

For each conductor i:
Width[i], Height[i], x[i], y[i], indice[i], resistivity[i], FillColor[i],
material_cond[i]

For each layer i:
Width_layer[i],   Height_layer[i],   x_layer[i],   y_layer[i],   e_layer[i],
s_layer[i], FillLayer[i]
```

Fig. 5.10: Sample for the ‘circuit’ file

5.2.4 Callback functions

It would be too extensive to describe all the possibilities given by the interface. Nevertheless, the principle described above is always valid. Each button pressed, each menu option selected, each action on the drawing area (resize, draw), will call an adequate callback function. This function will record some data, open additional windows if necessary, or update the structure on the drawing area. In order to determine which callback function is linked to which widget, the C-developer may look at the definition of the corresponding widget. Each object or widget is defined with certain characteristics including size, parameters, and callback function. The definition of a widget may be found in any of the following files.

The ‘main’ function of ‘frontend.c’ defines most of the widgets including all the menu options and all the toggle buttons. The file ‘graphtool.c’ contains the definition of the three forms located at the bottom of the layer column and the conductor column: ‘form_data_layer,’ ‘form_data_strip,’ ‘form_data_rect,’ and ’form_data_circle.’ It also defines the main drawing area and the forms ‘form_button’ and ‘form_button2’ including the four push-buttons ‘Plot,’ ‘Erase,’ ‘—>,’ and ‘<—’ for both conductors and layers. Finally, it includes the definition of the form ‘form_shape’ containing the four different shapes (strip, rectangle, circle, and polygon). The file ‘cust_dialog.c’ contains the definition of all the complex dialog windows except the help dialog, which is included in ’help_dialog.c.’

Once the name of the callback is defined, it is sufficient to look at the resource file ‘graphtool.h’ (see Appendix A), which contains all the functions of XRLGC ordered by file.

All the arrays and variables used in XRLGC are summarized in ‘graphtool.h’ as well. The C-file containing the callback function is hence easily found. For example, we shall consider the button ‘Erase’ located towards the bottom of the layer column. This button is part of the form ‘form_button2’ defined in the file ‘graphtool.c,’ more precisely in the function create_form_button2(). The definition of the button ‘Erase’ is given in Fig. 5.11. According to this definition, the callback of the button ‘Erase’ is the function ‘Erase_layer.’ However, this function does not ask any specific input (NULL). Nevertheless, if this button was selected with the help of the ‘context-sensitive help,’ the function ‘helpCB’ would be called with the input ‘---Erase.’

```
button4_l= XtVaCreateManagedWidget("Erase",
                                   xmPushButtonWidgetClass,form_button2,
                                   XmNtopAttachment,XmATTACH_WIDGET,
                                   XmNtopWidget, button3_l,
                                   XmNbottomAttachment,XmATTACH_FORM,
                                   XmNrightAttachment,XmATTACH_FORM,

                                   XmNleftAttachment,XmATTACH_FORM,
                                   NULL);

XtAddCallback(button4_l,XmNactivateCallback,Erase_layer,NULL);
XtAddCallback(button4_l,XmNhelpCallback,(XtCallbackProc)helpCB,
(XtPointer)"---Erase");
```

Fig. 5.11: Definition of a widget

In this example, the function ‘Erase_layer’ may be found in the resource file ‘graphtool.h’ (Appendix A). It is labeled as a function included in the file ‘layer_call.c.’ This function may therefore be easily found. Moreover, each function is preceded by a short overview of the content of the function in the form of comments embedded in the source. This method should be very useful for the C-developer in order to understand better and then improve the behavior of any widget.

5.3 Conclusion

This thesis has presented the program XRLGC at every level. First it includes the technical background and the methods adopted for the EM specialist. Second, its user guide is adapted for the beginner as well as for the computer expert. Finally, it includes an overview of the structure of XRLGC containing guidelines on how to approach and improve this complex program with relative ease. Ideally, this program should be preserved and adapted by

engineers who combine both the technical EM background and the ease of the good programmer.

Technical improvements to this program may involve the extension of the GUI to include another program that would perform the transient simulation of the multilayered, multiconductor structure with the input parameters *R*, *L*, *G* and *C*. An example of such a program and its user guide is given in [27]. Another possibility would be the addition of an optimization program involving for example neural networks. Moreover, the XRLGC program is implemented in C-Motif, complying to the industry standard X-Windows, thus escaping any hardware or software needs as well as any compatibility problems. However, a better choice might be to use the modern programing language Java, which is platform independent and may be called easily through the World Wide Web.

APPENDIX A
RESOURCE FILE OF XRLGC

The following file is the main resource file of XRLGC. It contains the definition of all the functions, arrays and variables used for the GUI.

```
/* ********************************************************************** *
 *
 *  graphtool.h
 *
 *  Headfile of XRLGC: Declarations of functions and variables
 *
 * ********************************************************************** */


/*-------------------------------------------------------------------------
-*/
/*--- Last Modified: Claire Lestrade, February 97         --*/
/*-------------------------------------------------------------------------
-*/



/*****************************/
/* Declaration of Functions */
/*****************************/


/* functions included in frontend.c */

void  Clear_drawing (Widget widget,XtPointer client_data, XtPointer
call_data);
void DrawShapes (Display* display, Window window,GC gc,int width,int height);
void drawCB(Widget widget,XtPointer client_data, XtPointer call_data);
void Exit_mesh(Widget w, XtPointer client_data, XtPointer call_data);
void store_data(Widget w, XtPointer client_data, XtPointer call_data);
void NewFile(Widget w, int client_data, XmSelectionBoxCallbackStruct
*call_data);
void OpenFile(Widget w, int client_data, XmSelectionBoxCallbackStruct
*call_data);
void store_data_layer(Widget w, XtPointer client_data, XtPointer call_data);
void toggleCB(Widget widget,
              XtPointer client_data,
              XtPointer call_data);
void read_data(char *fname);
void colour_call (int n,int p);
void cancelCB(Widget w, XtPointer client_data, XtPointer call_data);
void okCB(Widget w, XtPointer client_data, XtPointer call_data);
void AddStdCallbacks(Widget widget, char* name);
```

```
/* functions included in manageform.c */

void manage_strip(Widget widget,XtPointer client_data, XtPointer call_data);
void manage_rect(Widget widget,XtPointer client_data, XtPointer call_data);
void manage_circle(Widget widget,XtPointer client_data, XtPointer call_data);

/* functions included in graphtool.c */

void create_drawing_area(void);
void  create_form_strip(void);
void  create_form_circle(void);
void create_form_rect(void);
void create_form_button(void);
void create_form_button2(void);
void  create_type_frame(void);
void  create_graphic_context(void);
void  draw_cbk(Widget w, XButtonEvent *event,
               String *args, int *num_args);
void Equal (int i);


/*functions included in cond_call.c*/

void Erase_conductor(Widget w, XtPointer client_data, XtPointer call_data);
void New_conductor(Widget w, XtPointer client_data, XtPointer call_data);
void go_forward(Widget w, XtPointer client_data, XtPointer call_data);
void go_back(Widget w, XtPointer client_data, XtPointer call_data);


/*functions included in layer_call.c*/

void New_Layer_callback(Widget w, XtPointer client_data, XtPointer call_data);
void go_back_layer(Widget w, XtPointer client_data, XtPointer call_data);
void Erase_layer(Widget w, XtPointer client_data, XtPointer call_data);
void go_forward_layer(Widget w, XtPointer client_data, XtPointer call_data);


/*functions included in RLGC.c*/

void RLGC_main(void);

/*functions included in V2D.c*/

void V2D_main(double xb, double yb, double xt, double yt, int Nx, int Ny, int
Ne);

/*functions included in apply.c*/

void Solve(Widget w, XtPointer client_data, XtPointer call_data);
void RunMesh(Widget w, XtPointer client_data, XtPointer call_data);
void RunV2D(void);

/*functions included in cust_dialog.c*/
```

```
Widget CreateCustomDialog(Widget parent,char* name);
Widget CreateCustDlg(Widget parent);
Widget CreateMatrixDlg(Widget parent);
Widget CreateUnityDlg(Widget parent);
Widget CreateQualityDlg(Widget parent);
Widget CreateScaleDlg(Widget parent);

void scaleCB (Widget widget, XtPointer client_data, XtPointer call_data);

void manageCust (Widget widget, XtPointer client_data,XtPointer call_data);
void managematrix (Widget widget, XtPointer client_data,XtPointer call_data);
void manageunity (Widget widget, XtPointer client_data,XtPointer call_data);
void manageQuality (Widget widget, XtPointer client_data,XtPointer call_data);
void manageScale (Widget widget, XtPointer client_data,XtPointer call_data);
void manageVoltage (Widget widget, XtPointer client_data,XtPointer call_data);

void  create_graphic_context_result(void);
void DrawShapes2 (Display* display, Window window,GC gc,int width,int height);
void drawCB2(Widget widget,XtPointer client_data, XtPointer call_data);
void Get_Window_Callback(Widget widget,
             XtPointer client_data,
             XtPointer call_data);

/*functions included in help_dialog.c*/

void manageIntro (Widget widget, XtPointer client_data,XtPointer call_data);
Widget CreateIntroDlg(Widget parent);
void manageHelp (Widget widget, XtPointer client_data,XtPointer call_data);
Widget CreateIntroDlg(Widget parent);
void HelpDlgShow(void);
void HelpDlgCreate (Widget parent,
                    XtCallbackProc topic_callback,
                    XtPointer topic_data,
                    XtCallbackProc help_callback,
                    XtPointer help_text_data,
                    XtPointer help_topic_data);
void HelpDlgSetText(char* text);
void HelpDlgAddTopic(char* topic);
void HelpDlgDeleteAllTopics(void);
void helpCB (Widget widget,XtPointer client_data,XtPointer call_data);
void topicCB (Widget widget,XtPointer client_data,XtPointer call_data);
Widget CreatePushbuttonWithHelp(Widget parent,
                                char* name,
                                XtCallbackProc callback,
                                XtPointer client_data,
                                XtCallbackProc help_callback,
                                XtPointer help_data);
void contextCB(Widget widget, XtPointer client_data, XtPointer call_data);


/*functions included in toolbar.c*/

Pixmap CreatePixmapFromBitmap (Widget widget,
                               int weight,
```

```
                                int height,
                                Pixmap bitmap);
Boolean LoadBitmapLabel (Widget widget,char* filename);
Pixmap ReadBitmapFile(Widget widget,
                      char* filename,
                      int* width,
                      int* height);
void SetLabelPixmap(Widget widget, Pixmap pixmap);


/***********************************/
/* Global variables and constants */
/***********************************/


/*DEFINE*/

#define PI 3.14159265358979323846264 3
#define BUFFERSIZE 800
#define OK 1
#define CANCEL 2
#define MAXIMAL_NUMBER_SEGMENTS 20
#define NC 20  /* Maximal number of conductors*/
#define NS 10 /* Maximal number of layers*/
#define N_E 5 /*number of edge: circle*/
/*#define E 100  Scale (echelle)*/


/* Global variables*/

int type, compteur_s,compteur_r,compteur_c,compteur,nombre_conducteur,
compteur_local,type_local,compteur_local2;
int nombre_layer,compteur_layer;
int width_total,height_total;
Arg al[10],args[10];
Cardinal ac;
double size;
double ind_unity,capa_unity,res_unity,length_unity;
int no_exit,singular_matrix;
int E;

int Chosen_NB,N_F,N_F_w,N_F_h;


Widget toplevel,mainbox,form_cond,form_button,frame,label1,label2,label_s,
button1,button2,button3,labelx,labely,frame_shape,form_shape,label3,
labelx2,labely2,textex2,textey2,texte3,labelx3,labely3,button4,
label_layer,form_layer,form_common,Clear,Exit,menubar,menus,menus2,menus3,
menus4,cascades,cascades2,cascades3,cascades4,buttons[9],buttons1[3],
buttons2[5],buttons3[5],dialog1,dialog2,New,frame_cond,frame_layer,
radio_box,radio_button1,radio_button2,radio_button3,radio_button4,
frame_radio_box, radio_box2,radio_button12,radio_button22,radio_button32,
radio_button42,frame_radio_box2,labele,textee,labels,textes,form_es,
label1_s,texte1_s,label2_s,texte2_s,textex_s,textey_s,labelx_s,labely_s,
```

```
labele_s,labels_s,textee_s,textes_s,form_data_rect_s,New_layer,
radio_box_g,radio_button1_g,radio_button2_g,radio_button3_g,
radio_button4_g,frame_radio_box_g,cust,unity,RLGC,drawing_result,
label_matrixC,label_matrixL,label_matrixR,label_matrixCo,label_RLGC,sep2,
Intro,label_intro,title,warning,working,Quality,label_quality,
title_quality,scale_quality,Scale,texte_scale,Voltage,frame_Voltage,
label_Voltage,title_voltage,texte2_V,texte3_V,texte5_V,texte6_V,texte2_V2,
texte5_V2,texte2_V3;


XtAppContext app;
int k, n,i,screen,screen2;
int store_layer,store;
char buffer[BUFFERSIZE];
double Width[NC], Height[NC], x[NC],y[NC],X[NC],Y[NC],resistivity[NC];
int indice[NC];
long int FillColor[NC],FillLayer[NS];
double Width_layer[NS], Height_layer[NS],
x_layer[NS],y_layer[NS],e_layer[NS],s_layer[NS];
int current_type_layer;
int type_layer[NS];
int current_material_cond, current_material_strip, current_material_rect,
current_material_circle;
int material_cond[NC];
int ground,new_file;
char *name_file;
char input_file[256];
char output_file[256];
char Q2D_file[256];

Widget texte1,texte2,texte4,label4, label, textex, textey, textex3, textey3,
drawing;
Widget form,form_data_rect,form_data_strip,form_data_circle;
Widget strip,rectangle,polygon,circle;
Widget button1_l,button2_l,button3_l,button4_l,form_button2;

/*graphics*/

GC gc;
Display *display;
XGCValues xgcv;
unsigned long gc_mask;
Window window;
long int fill_pixel,fill_layer; /* stores current colour
                          of fill - black default */
Colormap cmap;
XtActionsRec actions;
int new;

char colours[256][9];


/*graphics for the result (Potential distribution)*/
```

```
GC gc2;
Display *display2;
XGCValues xgcv2;
unsigned long gc_mask2;
Window window2;
```

REFERENCES

[1] A. Ruehli, H. Heeb, "Circuit model for three-dimensional geometries including dielectrics," *IEEE Trans. Microwave Theory Tech.*, vol. 40, pp. 1507-1516, July 1992.

[2] J. E. Schutt-Aine, "Signal integrity and interconnects for high-speed applications," class notes for ECE 497-JS, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Fall 1997.

[3] A. Deutsch et al., "Modeling and characterization of long on chip interconnections for high performance microprocessors," *IBM J. Res. Develop.*, vol. 39, pp. 547-567, Sept. 1995

[4] J. E. Schutt-Aine, "Electromagnetic modeling of packages and interconnects," Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Tech. Rep., June 1997.

[5] K. S. Oh, J. E. Schutt-Aine, R. Mittra and W. Bu, "Computation of the equivalent capacitance of a via in a multilayered board using the closed-form Green's function," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-44, pp. 347-349, Feb. 1996.

[6] J. E. Schutt-Aine and K. S. Oh, "Modeling interconnections with nonlinear discontinuities [in MCMs]," *IEEE Int. Symp. on Circuits and Syst.*, vol. 3, pp. 2122-2124, 1993.

[7] J. E. Schutt-Aine, "Scattering parameters for the simulation of transmission line networks," *IEEE Int. Symp. on Circuits and Syst.*, vol. 4, pp. 2693-2695, 1990.

[8] W. T. Beyene and J. E. Schutt-Aine, "Accurate frequency domain modeling and efficient circuit simulation of high-speed packaging interconnects," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 1941-1947, Oct. 1997.

[9] W. T. Beyene and J. E. Schutt-Aine, "Efficient transient simulation of high-speed interconnects characterized by sampled data," in *Proc. IEEE 5th Topic. Meeting on Electric. Perform. of Electronic Packag.*, Oct. 1996, pp. 162-165.

[10] D. B. Kuznetsov and J. E. Schutt-Aine, "The optimal transient simulation of transmission lines," *IEEE Trans. Circuits Syst.-I*, vol. CAS-43, pp. 110-121, Feb. 1996.

[11] T. Dhaene, L. Martens, and D. De Zutter, "Transient simulation of arbitrary nonuniform interconnection structures characterized by scattering parameters," *IEEE Trans. Circuits Syst.*, vol. 39, pp. 928-937, Nov. 1992.

[12] D. B. Kuznetsov, J. E. Schutt-Aine and R. Mittra, “The optimal transient simulation of distributed lines,” *IEEE Multichip Module Conf.*, pp. 164-169, Feb. 1995.

[13] J. S. Roychowdhury and D. O. Pederson, “Efficient transient simulation of lossy interconnects,” *in Proc. 28Th ACM/IEEE Design Automation Conf.,* June 1991, pp. 740-745.

[14] N. Orhanovic and V. K. Tripathi, “Nonlinear transient analysis of coupled RLGC lines by the method of characteristics,” *Int. J. Microwave Millimiter-Wave CAE,* vol. 2, pp. 109-115, 1992.

[15] E. Chiprout and M. S. Nakhla, “Transient waveform estimation of high-speed MCM networks using complex frequency hoping,” in *Proc. Multichip Module Conf. (MCMC),* Santa-Cruz, CA, March 1993, pp.134-139.

[16] D. B Kuznetsov and J. E. Schutt-Aine, “Efficient circuit simulation of nonuniform transmission lines,” submitted to *IEEE Trans. Microwave Theory Tech.,* Jan. 1997.

[17] F.-Y. Chang, “Transient simulation of frequency-dependent nonuniform coupled lossy transmission lines,” *IEEE Trans. Components, Packaging and Manufacturing Tech.,* vol. 17, pp. 3-14, Feb. 1994.

[18] K. S. Oh and J. E. Schutt-Aine, “Efficient modeling of interconnects and capacitive discontinuities in high-speed digital circuits,” *EMC Lab Technical Report*, no. EMC 95- 1, June 1995.

[19] O. Nayac, G. Irwin, and A. Neufeld, “GUI enhances electromagnetic transients simulation tools,” *IEEE Computer Applications in Power*, vol. 8, pp. 17-22, Jan. 1995.

[20] K. S. Oh, D. B. Kuznetsov and J. E. Schutt-Aine, “Capacitance computation in a multilayered dielectric medium using closed-form spatial Green’s function,” *IEEE Trans. Microwave Theory Tech.*, vol. MTT-42, pp. 1443-1453, Aug. 1994.

[21] I. Hajj, “Computational techniques for circuit analysis and design,” class notes for ECE 452, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Spring 1997.

[22] W. T. Weeks, “Calculation of coefficients of capacitance of multiconductor transmission lines in the presence of a dielectric interface,” *IEEE Trans. Microwave Theory Tech.*, vol. MTT-18, pp. 35-43, Jan. 1970.

[23] C. Wei, R. F. Harrington, J. R. Mautz, and T. K. Sarkar, "Multiconductor transmission lines in a multilayered dielectric media," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-32, pp. 439-450, April 1984.

[24] W. Delbare and D. D. Zutter, "Space-domain Green's function approach to the capacitance calculation of multiconductor lines in multilayered dielectrics with improved surface charge modeling," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-37, pp. 1562-1568, Oct. 1989.

[25] S. Papatheodorou, R. F. Harrington, and J. R. Mautz, "The equivalent circuit of a microstrip crossover in a dielectric substrate," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-38, pp. 135-140, Feb. 1990.

[26] M. I. Aksun and R. Mittra, "Derivation of closed-form Green's functions for a general microstrip geometry," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-40, pp. 2055-2062, Nov. 1992.

[27] J. Schutt-Aine, *Transmission Line Simulation Program nln.c*. User guide, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Nov. 1997.