FastHenry USER'S GUIDE

Version 3.0

M. Kamon L.M. Silveira C. Smithhisler J. White

Research Laboratory of Electronics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139 U.S.A.

11 November 1996

This work was supported by Defense Advanced Research Projects Agency contract N00014-91-J-1698, a National Science Foundation Graduate Fellowship, and grants from IBM and Digital Equipment Corporation.

Contents

This manual describes FastHenry, a three-dimensional inductance extraction program. FastHenry computes the frequency dependent self and mutual inductances and resistances between conductors of complex shape. The algorithm used in FastHenry is an acceleration of the mesh formulation approach. The linear system resulting from the mesh formulation is solved using a generalized minimal residual algorithm with a fast multipole algorithm to efficiently compute the iterates. See Appendix A for references.

This manual is divided into four sections. The first section explains the syntax for preparing input files for FastHenry. The input files contain the description of the conductor geometries. The second section describes how to run the program and process the output. The third section describes the generation of postscript images to visualize the three dimensional geometries defined in the input file. The fourth section shows how to use Matlab to observe current distribution in reference plances.

The files "nonuniform_manual_*.ps" constitute an important supplement to this manual and describe the new routines for specifying a nonuniformly discretized reference plane.

Information on compiling FastHenry, obtaining the FastHenry source code, producing this document, and corresponding about FastHenry is given in Appendix A.

Appendix B summarizes most of the changes in this version, 3.0, over the previous version, 2.5.

1 How to Prepare Input Files

This section of the manual describes how to prepare input files for FastHenry. The input files specify the discretization of conductor volumes into filaments. The input file specifies each conductor as a sequence of straight segments, or elements, connected together at nodes. Each segment has a finite conductivity and its shape is a cylinder of rectangular cross section of some width and height. A node is simply a point in 3-space. The cross section of each segment can then be broken into a number of parallel, thin filaments, each of which will be assumed to carry a uniform cross section of current along its length. The first two parts of this section describe the file format through simple examples. A detailed description is in the third part, and more complex examples can be found in the last section and later in the manual.

1.1 A Simple Example

The following is an input file which calculates the loop inductance of four segments nearly tracing the perimeter of a square. It is described more thoroughly on the next page.

```
**This is the title line. It will always be ignored**.
* Everything is case INsensitive
* An asterisk starts a comment line.

* The following line names millimeters as the length units for the rest
* of the file.
.Units MM
```

```
* Make z=0 the default z coordinate and copper the default conductivity.
* Note that the conductivity is in units 1/(mm*Ohms), not 1/(m*Ohms)
* since the default units are millimeters.
.Default z=0 sigma=5.8e4

* The nodes of a square (z=0 is the default)
N1 x=0 y=0
N2 x=1 y=0
N3 x=1 y=1
N4 x=0 y=1
N5 x=0 y=0.01

* The segments connecting the nodes
E1 N1 N2 w=0.2 h=0.1
E2 N2 N3 w=0.2 h=0.1
E3 N3 N4 w=0.2 h=0.1
E4 N4 N5 w=0.2 h=0.1

* define one 'port' of the network
.external N1 N5

* Frequency range of interest.
.freq fmin=1e4 fmax=1e8 ndec=1

* All input files must end with:
.end
```

As described in the comments, `.Units MM` defines all coordinates and lengths to be in millimeters. All lines with an `N` in the first column define nodes, and all lines starting with `E` define segments. In particular, the line

```
E1 N1 N2 w=0.2 h=0.1
```

defines segment E1 to extend from node `N1` to `N2` and have a width of 0.2 mm and height of 0.1 mm as drawn in Figure 1. If the $n \times n$ impedance matrix, $Z(\omega)$, for an n-conductor problem is thought of as the parameters describing an n-port network, then the line

```
.external N1 N5
```

defines `N1` and `N5` as one port of the network. In this example, only one port is specified, so the output will be a 1×1 matrix containing the value of the impedance looking into this one port.

FastHenry calculates $Z(\omega)$ at the discrete frequencies described by the line

```
.freq fmin=1e4 fmax=1e8 ndec=1
```

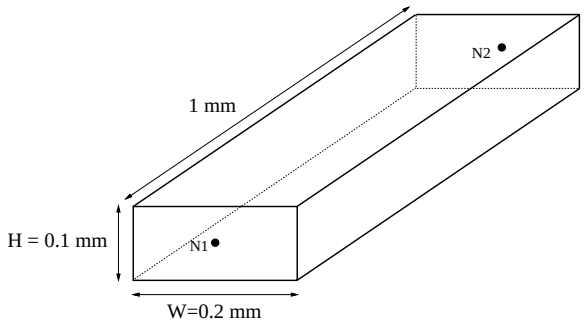

Figure 1: Example Segment for Sample Input File

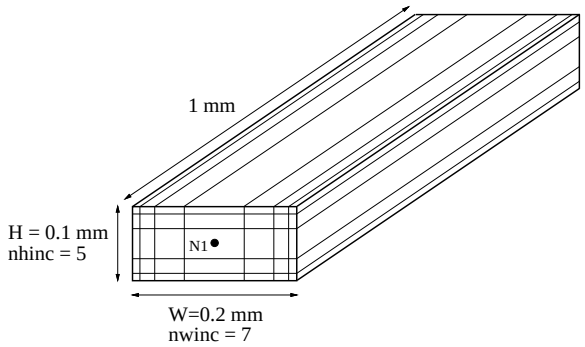

Figure 2: Segment discretized into 35 filaments

where `fmin` and `fmax` are the minimum and maximum frequencies of interest, and `ndec` is the number of desired frequency points per decade. In this case, $Z(\omega)$ will be calculated at $10^4, 10^5, 10^6, 10^7$, and 10^8 Hz. All input files must end with `.end`.

In the above example, FastHenry created one filament per segment since no discretization of the segments into filaments was specified. In order to properly model non-uniform cross sectional current due to skin and proximity effects, a finer discretization must be used. Finer filaments are easily specified in the segment definition. For example, replacing the definition for `E1` with

```
E1 N1 N2 w=0.2 h=0.1 nhinc=5 nwinc=7
```

specifies that `E1` is to be broken up into thirty-five filaments: five along its height (`nhinc=5`) and seven along its width (`nwinc=7`). See Figure 2.

1.2 Another Simple Example

To continue teaching by example, what follows is an example of computing the loop inductance of an L shaped trace over a ground plane with the trace's return path through the plane as shown in Figure 3. Note that a line beginning with '+' is a continuation of the previous line.

```
* A FastHenry example using a reference plane
```

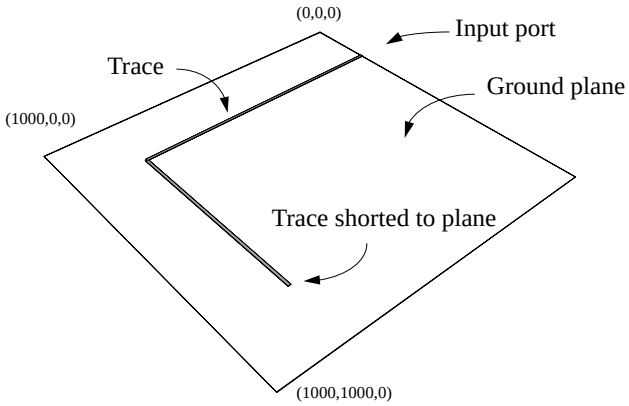

Figure 3: L-shaped trace over ground plane

```
*  This example defines an L shaped trace above a ground plane which
*  has a return path through the plane

*  Set the units for all following dimensions
.units mils
*
* Define ground plane and nodes to reference later
*
*  First define 3 corners of the plane, in clockwise or counter-cw order
g1 x1 = 0     y1 = 0     z1 = 0
+  x2 = 1000  y2 = 0     z2 = 0
+  x3 = 1000  y3 = 1000  z3 = 0
*     thickness:
+  thick = 1.2
*     discretization:
+  seg1 = 20 seg2 = 20
*     nodes to reference later:
+  nin  (800,800,0)
+  nout (0,200,0)

* Some defaults to model skin effects
.default nwinc=8 nhinc=1
*
* L shaped trace over ground plane
*
* The nodes
N1 x=0 y=200 z=1.5
N2 x=800 y=200 z=1.5
N3 x=800 y=800 z=1.5

* The elements connecting the nodes
```

```
E1 N1 N2 w=8 h=1
E2 N2 N3 w=8 h=1

* Short together the end of the L shaped trace (N3) and its corresponding
*  point on the ground plane directly beneath (nin)
.equiv nin n3

* compute loop inductance from beginning of L (N1) to its corresponding
*   point directly underneath (nout)
.external N1 nout

* Compute impedance matrix for one very low frequency (essentially DC)
*   and one very high frequency
.freq fmin=1e-1 fmax=1e19 ndec=0.05

* mark end of file
.end
```

In this example, the ground plane is a $1000mil \times 1000mil$ sheet of copper (by default) defined by three of its four corners, $(0,0,0)$, $(1000,0,0)$, $(1000,1000,0)$. The plane is 1.2 mils thick and its discretization is specified by `seg1` and `seg2` (see Section 1.3.9). The discretization of the plane forms a grid of nodes and interconnecting segments. `nin` and `nout` refer to the internal nodes of the plane which are closest to $(800,800,0)$ and $(0,200,0)$, respectively.

To model skin effects on the trace, the default for the number of filaments per segment is set to 8.

To compute loop inductance, the node at one end of the trace is shorted to the plane by declaring the the node to be "electrically equivalent" to the node directly underneath.

```
.equiv nin n3
```

and then the other end is declared as the "port" with the `.external` statement.

In this case, a single loop impedance will be computed. If, however, the `.equiv` and `.external` were replaced with

```
.external N1 N3
.external nout nin
```

then the partial inductances and resistances of these two paths would be computed yielding a 2×2 impedance matrix.

Computing the impedance for only two frequencies is useful for visualization of the distribution of current in the reference plane as described later in Section 4.

1.3 Input File Syntax

The previous section described many of the basics required for an input file. This section gives a more complete and detailed description of the input file format and should serve as a reference.

Some general facts about file syntax:

- Lines are processed sequentially.
- Uppercase is converted to lower case.
- "*" at the beginning of a line marks a comment line.
- Lines are restricted to 1000 characters but can be continued with a "+" as the first character of subsequent lines. Intervening "*" lines are allowed in this release.
- The first line in the file is considered the title line and is ignored. It is recommended that this line start with an "*" for future compatibility and file concatenation.
- The file must end with the `.End` keyword.
- Names of objects are limited to 80 characters.

In general, each line of the input file will either define some geometrical object, such as a node or segment, or it will specify some program parameter. All input lines that define geometrical objects begin with a letter defining their type, and then some unique alphanumeric string of up to 80 characters. For instance, all node definitions begin with the letter `N`. This sets object lines apart from parameter specification lines which begin with a period, ".".

The remainder of this section will describe all possible input lines. In the following description, any argument enclosed in '[ ]' indicates an optional argument. If not included on the input line, the actual value used for this argument will be either the program default, or the user default defined by the `.Default` keyword (described below).

1.3.1 Node Definitions

```
Syntax:  Nstr [x = x_val] [y = y_val] [z = z_val]
```

This defines a node called `Nstr` where `str` is any alphanumeric string. The first character on the line must begin with an `N` for this to be interpreted as a node definition. The node will have location (x_val, y_val, z_val) where each coordinate has units defined by the `.Units` keyword.

Any of the coordinates can be omitted assuming that a default value has been previously specified with the `.Default` keyword. Otherwise, an error will occur and the program will exit.

1.3.2 Segment Definitions

```
Syntax:  Estr node1 node2 [w = value] [h = value] [sigma, rho = value]
                 [wx = value wy = value wz = value]
                 [nhinc = value] [nwinc = value] [rh = value] [rw = value]
```

This defines a segment called `Estr` where `str` is any alphanumeric string. The first character on the line must begin with the letter `E` for this to be interpreted as a segment definition. The segment will extend from node `node1` to node `node2` which must be previously defined node names. `h` and `w` are the segment height and width. Either `sigma`, the conductivity, or `rho`, the resistivity, can be specified for the segment.

Discretization of the segment into multiple, parallel thin filaments is specified with the `nhinc` and `nwinc` arguments. `nhinc` specifies the number of filaments in the height direction, and `nwinc`, the number in the width direction. Both must be integers. See Figure 2. By default, the ratio of adjacent filaments is 2.0 as in Figure 2, however this can changed with the `rh` and `rw` arguments which specify the ratio in the height and width direction, respectively. While an automatic technique of dividing a section into filaments based on the skin depth may be a better approach, no such feature is available. See Section 2.4 for guidance in choosing filament discretizations.

To specify the orientation of the cross section, `wx`, `wy`, and `wz` represent any vector pointing along the width of the segment's cross section. If these are omitted, the width vector is assumed to lie in x-y plane perpendicular to the length. If the length direction is parallel to the z-axis, then the width is assumed along the x-axis.

`h` and `w` can be omitted provided they are assigned a default value in a previous `.Default` line.

`nhinc`, `nwinc`, and `sigma` or `rho` can be omitted, and if not previously given a default value, then 1, 1, and the conductivity of copper, respectively, are used as default values.

Note that the nodes used to define the elements must be defined under **Node Definitions** described above and cannot be reference plane nodes (See Section 1.3.9 for a description of reference planes). To connect to a reference plane, the user must instead create a new node at the desired location as described in **Node Definitions** and then use the `.Equiv` keyword to equivalence the reference plane node and the new node. For instance, the following is not allowed:

```
g1 x1 = 0     y1 = 0     z1 = 0
+  x2 = 1000  y2 = 0     z2 = 0
  .
  .
+ n_gp (5,5,0)

N1 x=5 y=5 z=10
E1 N1 n_gp
```

The legal way to define the segment would be

```
g1 x1 = 0     y1 = 0     z1 = 0
+  x2 = 1000  y2 = 0     z2 = 0
```

```
 .
 .
+ n_gp (5,5,0)

N1 x=5 y=5 z=10
N2 x=5 y=5 z=0
E1 N1 N2
.equiv N2 n_gp
```

1.3.3 `.Units` keyword

```
Syntax:  .Units unit-name
```

This specifies the units to be used for all subsequent coordinates and lengths until the end of file or another `.Units` specification is encountered. Allowed units are kilometers, meters, centimeters, millimeters, micrometers, inches, and mils with `unit-name` specified as `km, m, cm, mm, um, in, mils`, respectively.

Note that this keyword affects the expected units for the conductivity and resistivity.

1.3.4 `.Default` keyword

```
Syntax:  .Default [x = value] [y = value] [z = value] [w = value]
                  [h = value] [sigma, rho = value]
                  [nhinc = value] [nwinc = value] [rh = value] [rw = value]
```

This keyword specifies default values to be used for subsequent object definitions. A certain default value is used until the end of the file, or until it is superseded by another `.Default` line changing that value.

1.3.5 `.External` keyword

```
Syntax:  .External node1 node2 [portname]
```

This keyword specifies node `node1` and node `node2` as a terminal pair or port whose impedance parameters should be calculated for the output impedance matrix. If an input file includes n `.External` lines, then the impedance matrix will be an $n \times n$ complex matrix. The port can be given a name by specifying a single word string `portname`. This name is used to reference this port in the output file, `Zc.mat`, and is also used with the command line `-x` option described in Section 2.3.

This keyword effectively places a voltage source between these nodes and will later use the current through that source to determine an entry in the admittance matrix. The first node specified is the positive node. Note that it is up to the user to insure that there are NO loops of only voltage sources. Also, a voltage source with no possible return path will always carry zero current producing a row of zeros in the admittance matrix. The output impedance matrix will thus be nonsense (NaN).

1.3.6 .Freq keyword

```
Syntax:  .Freq fmin=value fmax=value [ndec = value]
```

This keyword specifies the frequency range of interest. `fmin` and `fmax` are the minimum and maximum frequencies of interest, and `ndec` is the number of desired frequency points per decade. FastHenry must perform the entire solution process for each frequency.

Note that `ndec` need not be an integer. For instance,

```
.freq fmin=1e3 fmax=1e7 ndec=0.5
```

will have FastHenry calculate impedance matrices for $f = 10^3, 10^5$, and 10^7 Hz.

If `fmin` is zero, FastHenry will run only the DC case regardless of the value of `fmax`.

1.3.7 .Equiv keyword

```
Syntax:  .Equiv node1 node2 node3 node4 ...
```

This keyword specifies that nodes `node1, node2, node3, node4,...` are to be considered electrically equivalent yet maintain their separate spatial coordinates. The nodes are effectively 'shorted' together. If any of the node names are not previously defined, then they become pseudonyms for those nodes in the list which are defined. Note that the current flow between equivalenced nodes is not electromagnetically modeled.

1.3.8 .End keyword

```
Syntax:  .End
```

This keyword specifies the end of the file. All subsequent lines are ignored. This line must end the file.

1.3.9 Reference Plane definitions

Note: It is recommended that nonuniformly discretized planes be used instead of what is described here if possible. See the document "nonuniform_manual_*.ps". Uniform planes described below can still be used if they satisfy your needs. Also, for new users, understanding uniform planes is necessary to understand the nonuniform_manual.ps document.

Uniformly discretized planes

```
Syntax:  Gstr x1=value y1=value z1=value x2=value y2=value z2=value
              x3=value y3=value z3=value
              thick=value seg1=value seg2=value
              [segwid1 = value] [segwid2 = value]
              [sigma, rho = value]
              [nhinc=value] [rh=value]
              [relx=value] [rely=value] [relz=value]
              [Nstr1     (x_val,y_val,z_val) ]
```

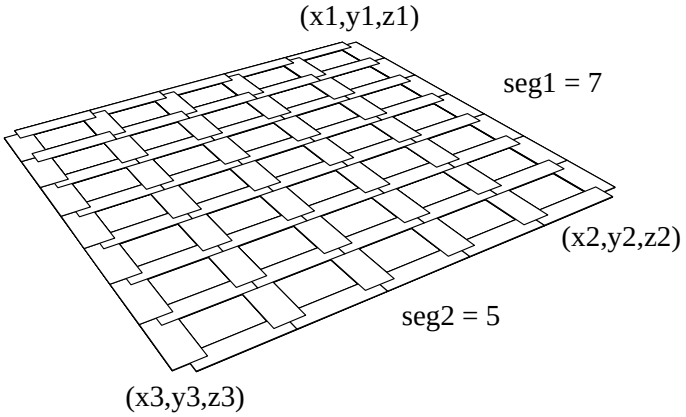

Figure 4: Discretization of a Reference Plane. Segments are one-third actual width.

```
[Nstr2     (x_val,y_val,z_val) ]
[Nstr3     (x_val,y_val,z_val) ].....
[hole <hole-type> (val1,val2,....)]
[hole <hole-type> (val1,val2,....)].....
```

This defines a uniformly discretized reference plane of finite extent and conductivity called `Gstr` where `str` is any alphanumeric string. The first character on the line must begin with the letter `G` for this to be interpreted as a reference (originally "ground") plane definition. The three locations $(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)$ mark three of the four corners of the plane in either clockwise or counterclockwise order. FastHenry will determine the fourth corner assuming the first three are corners of a rectangle. The thickness of the plane is specified with the `thick` argument and the conductivity with either the `sigma` or `rho` argument. Note that current is modeled to flow two dimensionally, i.e., not in the thickness direction.

The reference plane is approximated by first laying down a grid of nodes, and then with segments, connecting every node to its adjacent nodes excluding diagonally adjacent nodes. Each segment is given a height equal to the specified thickness of the plane and, by default, width equal to the node spacing. This choice of width completely fills the space between segments. Figure 4 shows a sample reference plane with segments that are one-third full width for illustration.

`seg1` and `seg2` specify the number of segments along each edge of the plane. `seg1` is the number of segments along the edge from $(x1, y1, z1)$ to $(x2, y2, z2)$ and `seg2`, the number along the edge from $(x2, y2, z2)$ to $(x3, y3, z3)$. Thus the total number of nodes created will be (`seg1`+1)*(`seg2`+1), and the total number of segments created will be $(\texttt{seg1} + 1) * \texttt{seg2} + \texttt{seg1} * (\texttt{seg2} + 1)$. Note that the line from the node at $(x1, y1, z1)$ to the node at $(x2, y2, z2)$ define the edge of the plane, however segments that run from $(x1, y1, z1)$ to $(x2, y2, z2)$ will overhang the edge by half their width since nodes are defined at the center of segments. This slight overapproximation to the width of the plane is avoided in the nonuniform discretization code described in `nonuniform_manual_*.ps`.

`nhinc` and `rh` can be used to specify a number of filaments for discretization of each segment along the thickness (See Section 1.3.2). This could be used for modeling

nonuniform current across the thickness. If `nhinc` is omitted, the value of 1 is used regardless of the `.Default` setting. If `rh` is omitted, the default value is used.

Note: If you do not want to use segwid1 and segwid2 or holes as described below, you can use the nonuniform plane definition as described in "nonuniform_manual.ps".

Connections to the plane

Since the reference plane nodes are generated internally, there is no way to refer to them later in the input file. The exceptions to this are the nodes explicitly referenced in the reference plane definition. The argument

```
Nstr1     (x_val,y_val,z_val)
```

where `str1` is an alphanumeric string, will cause all subsequent references to `Nstr1` to refer to the node in the plane closest to the point (x_val, y_val, z_val). Note that *no spaces* are allowed between the '()'. The referencing is accomplished internally by effectively doing

```
.Equiv Nstr1 <internal-node-name>
```

where `<internal-node-name>` is the internal node name of the nearest reference plane node.

If one or more of `relx`, `rely`, and `relz` are specified, then the above node referencing instead chooses the node closest to $(x\_val + relx, y\_val + rely, z\_val + relz)$. In other words, `relx`, `rely`, and `relz` default to 0 if not specified.

A coarsely discretized plane (small `seg1`, `seg2`) may cause two different node references to refer to the same reference plane node. FastHenry will warn of such an event, but it is not an error condition.

Note: It is recommended that connections to the plane be done with the "contact equiv_rect" utility for nonuniformly discretized planes as described in "nonuniform_manual.ps".

Meshed Planes

By default, the width of the segments of the plane are chosen to fill the space between adjacent segments. However, to model meshed planes the width of the segments can be chosen to be smaller with the `segwid1` and `segwid2` arguments. This gives the plane the appearance of a "mesh" as shown is Figure 4. `segwid1` specifies the width of the segments that are in the direction parallel to the plane edge from $(x1, y1, z1)$ to $(x2, y2, z2)$. `segwid2` specifies the width along the edge from $(x2, y2, z2)$ to $(x3, y3, z3)$.

Note that if the section of conductor between meshes is large compared to the size of the mesh holes, this method of discretization may be too coarse. For instance, assume the mesh consists of only nine holes on a 3x3 grid. In such a situation, consider using the `hole` functions described below to make the specific holes on a plane with `seg1,seg2` $>>$ 3. See also Section 1.4.3.

Holes in the plane

Holes can be specified in the plane with

```
hole <hole-type> (val1,val2,val3,...)
```

where `<hole-type>` is the hole type and the `valn`'s make the list of arguments to be sent to the hole generating function. Holes are generated by first removing reference plane nodes, and then removing all segments connected to those nodes. The following describes the available hole generating functions:

```
hole point (x,y,z)
```

removes the node nearest to the point (x,y,z).

```
hole rect (x1,y1,z1,x2,y2,z2)
```

removes a rectangular region whose opposite corners are the nodes in the plane nearest (x1,y1,z1) and (x2,y2,z2).

```
hole circle (x,y,z,r)
```

removes the nodes contained within the circle of radius `r` centered at (x,y,z).

```
hole user1 (val1,val2,...)
```

calls the user defined function hole_user1() to remove nodes. user1 - user7 are available. The user can add the functions to the source file `hole.c` contained in the release. See the functions hole_rect(), hole_point(), and hole_circle() to see examples of the format for writing user hole functions.

Any shaped hole can be formed with a combination of hole directives. A few exceptions exist, however. Forming a hole that isolates or nearly isolates a section of the plane is not allowed. FastHenry warns of this with:

```
Warning: Multiple boundaries found around one hole region
  possibly due to an isolated or nearly isolated region of conductor.
  This may lead to no unique solution.
```

If a nearly isolated section of conductor is truly desired, consider defining two separate planes and connecting them.

Holes may not be produced as expected if the discretization of the plane is coarse. It is recommended that the plane be viewed by using the options "`-f simple -g on`" options to generate a `zbuf` file which can be used to generate a postscript image of the plane as described in Section 3.

1.4 Other Examples

This section describes other examples using more of the features of FastHenry. Some examples were created by the authors and some contributed from users. All files are available with the FastHenry source code.

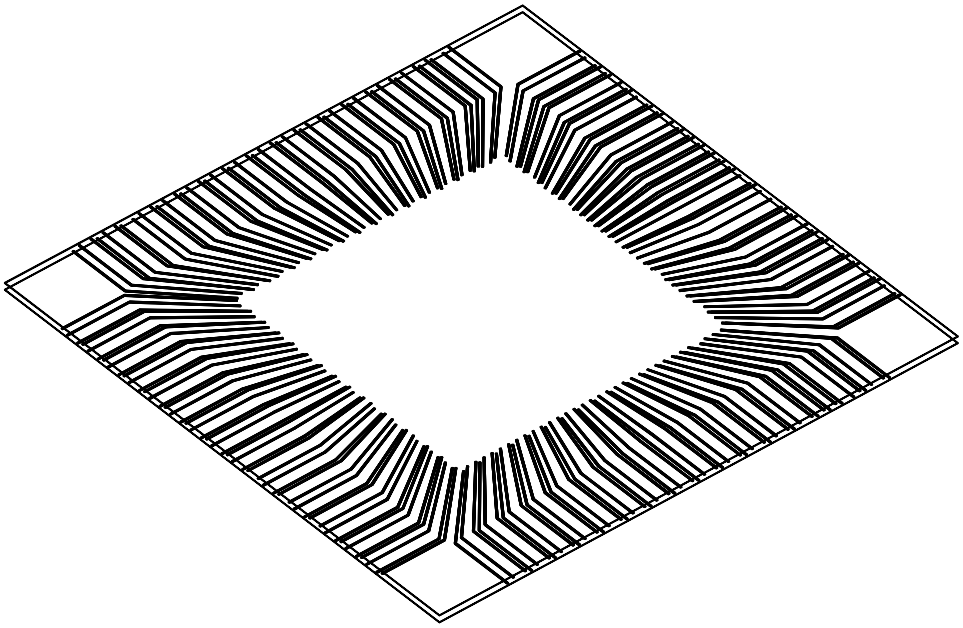

Figure 5: Printed Circuit Board example

1.4.1 Printed Circuit Board

The example file `gpexamp_copper.inp` shown in Figure 5 is a portion of a printed circuit board from Digital Equipment Corporation. This structure is to be placed underneath a PGA package. This geometry includes 2 reference planes sandwiching a large number of copper lines which lead to the center where the package would be placed. The set of copper lines are grouped into 18 groups according to functionality (see the many .equiv statements at the bottom of the file).

1.4.2 Vias and Meshed Planes

The example file `vias.inp` shown in Figure 6 shows three vias passing through a meshed ground plane. To form the mesh structure, the standard reference plane definition is used with the `segwid1` and `segwid2` arguments. Some of the dimensions for this example were taken from B.J. Rubin, "An electromagnetic approach for modeling high-performance computer packages," *IBM J. Res. Dev.*, Vol. 34, No. 4, July 1990.

1.4.3 Holey ground plane

The example file `holey_gp.inp` shown in Figure 7 is a punctured version of the reference plane example of section 1.2. The holes of the plane are formed with the `rect` and `circle` hole directives.

1.4.4 Pin-connect

The example file `pin-connect.inp` shown in Figure 8 is thirty-five pins of a 68-pin cerquad pin package from Digital Equipment Corporation.

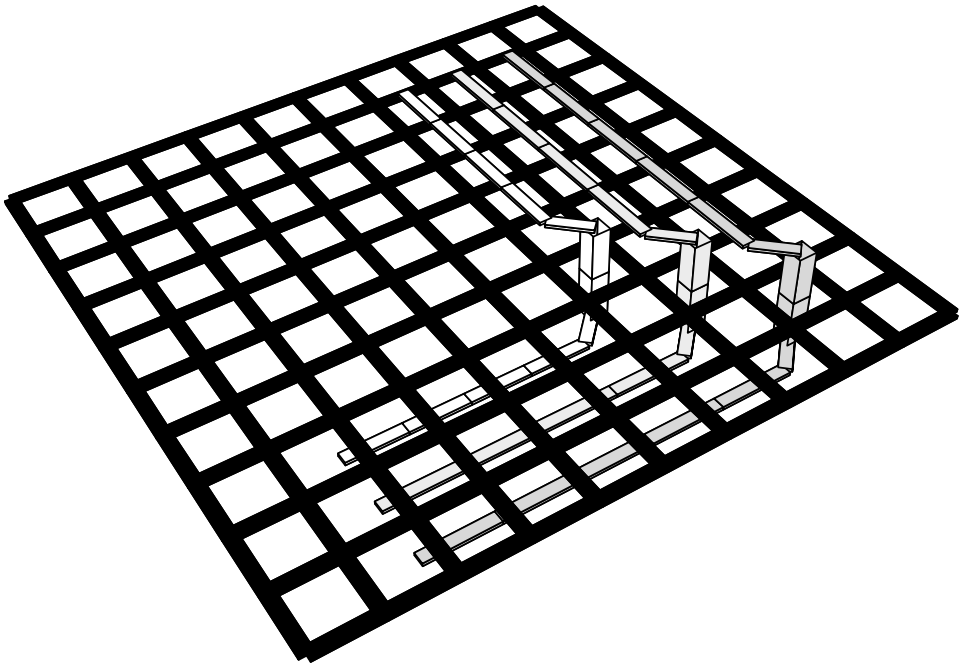

Figure 6: Vias through a meshed ground plane

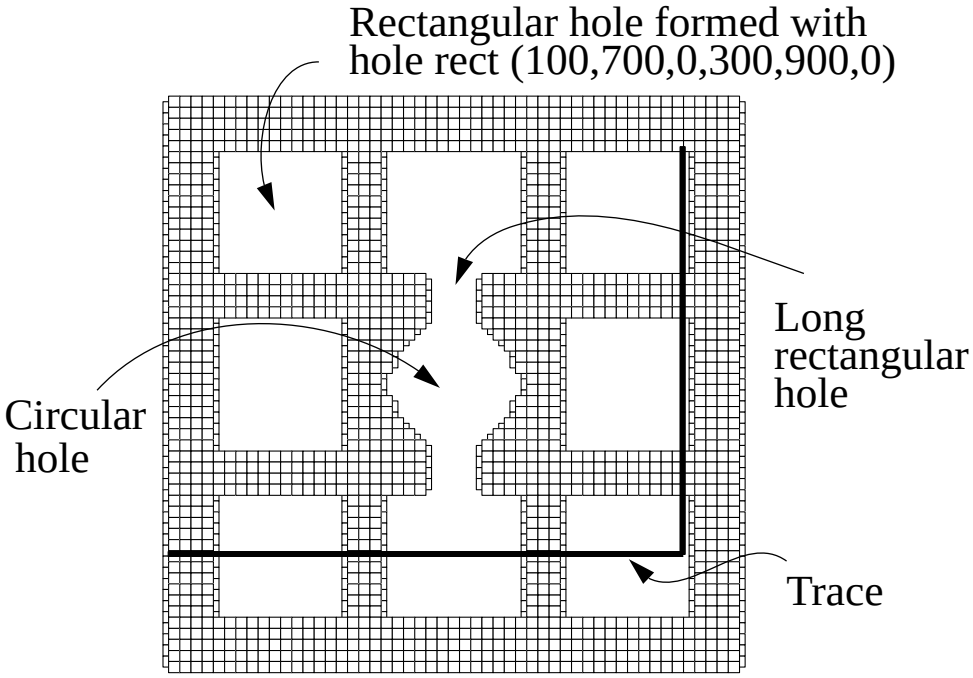

Figure 7: Trace over Ground Plane with holes

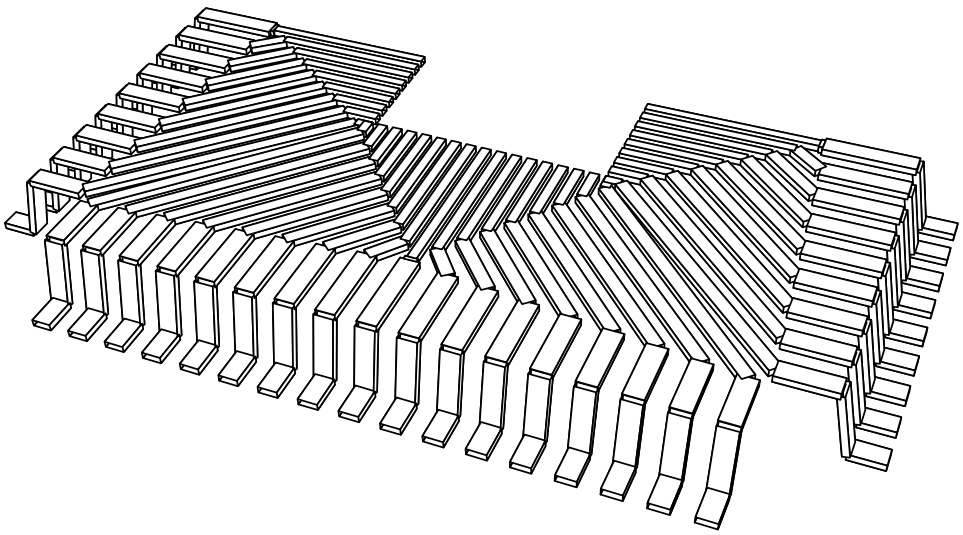

Figure 8: Thirty-Five Pins of a 68 pin Cerquad Package

1.4.5 Right Angle Connector

The example file `30pin.inp` from Teradyne Connection Systems Inc. shown in Figure 9 is a PCB right angle connector plugged into its thirty corresponding pins. The V-shaped clamping portion and right angle bend make up the connector as shown in the single pin illustration of Figure 10. In reality, the pin protrudes through the V-shaped clamp, but that section conducts no current and is not included.

Each of the thirty pins is a different shade of gray. The white portions of the V are also part of each pin but appear white due to the limits of the shading algorithm.

1.4.6 Photodetector

The example file `msm.inp` shown in Figure 11 is a metal-semiconductor-metal photodetector from NASA Langley Research Center and University of Virginia. These devices consist of a set of closely spaced, interdigitated electrodes deposited on an optically active semiconductor layer. Every other electrode is biased to the same potential, so that any two adjacent electrodes are at opposite electrostatic potentials. Becuase the electrodes are closely spaced, the electric fields between them are quite strong with application of low to moderate bias voltages. As a result, any carriers generated by absorption of light between the fingers are transported to the electrodes extremely rapidly giving fast response to short pulses of light.

2 Running FastHenry

The basic form of the FastHenry program command line is

```
fasthenry [<input file>] [<Options>]
```

Usually only the `input file` is specified. For example, the command

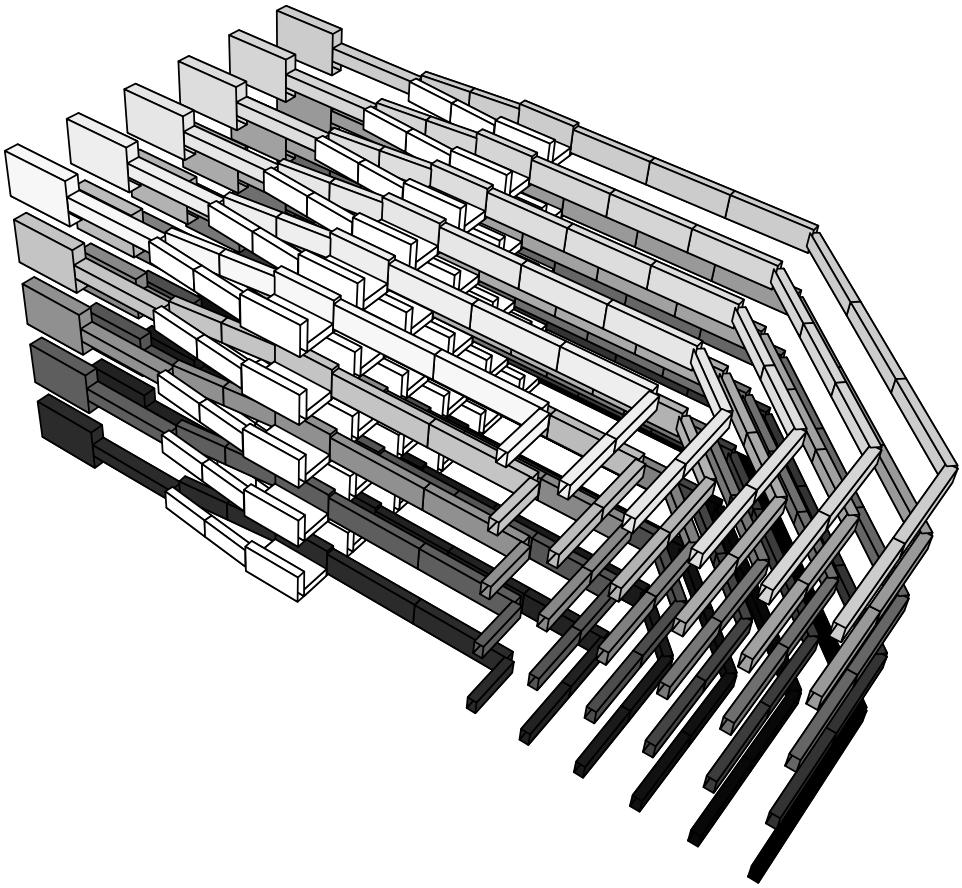

Figure 9: Thirty pin right angle connector

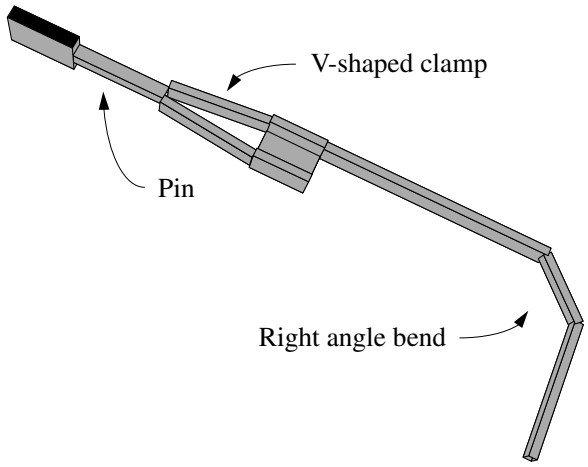

Figure 10: One pin of the thirty pin right angle connector

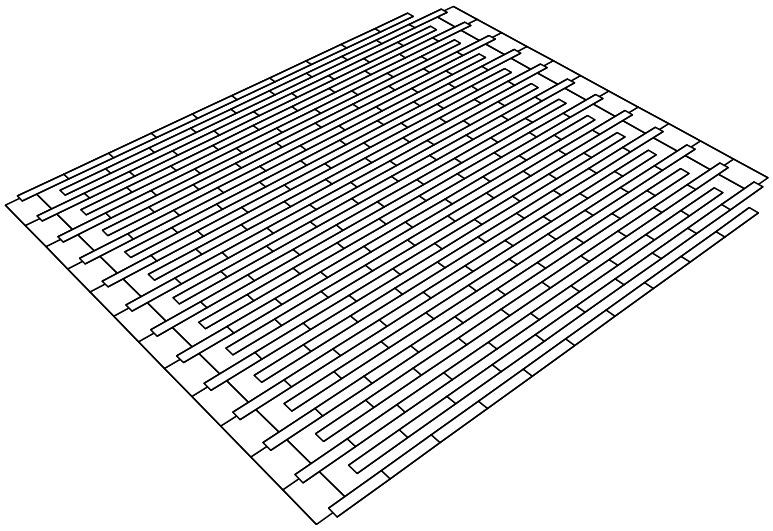

Figure 11: Metal-semiconductor-metal Photodetector

```
fasthenry pin-connect.inp
```

runs `fasthenry` on the example pin connect structure.

Information about the input file and other FastHenry information are sent to the standard output. The impedance matrices for the frequencies specified in the input file will be placed in the text file `Zc.mat`. This file also lists the correspondence between columns in the impedance matrix and the ports specified in the input file. The source file `ReadOutput.c` described in Section 2.2 is provided as a sample program for reading the output file for postprocessing.

2.1 Example Run

This section contains a sample run of FastHenry for the 30 pin connector example, `30pin.inp` shown in Figure 9. Comments describing the output appear along the right margin in addition to the `FastHenry` command line option which would change the described setting.

```
prompt % fasthenry 30pin
Running FastHenry 2.0 (10Aug94)
  Date: Wed Aug 10 16:31:35 1994
  Host: lyapunov
Solution technique: ITERATIVE                  <-- use GMRES (-s)
Matrix vector product method: MULTIPOLE        <-- and multipole algorithm (-m)
  Order of expansion: 2                        <-- order of multipole (-o)
Preconditioner: ON                             <-- GMRES preconditioner (-p)
Error tolerance: 0.001                         <-- relative tol (-t, -b)
Reading from file: 30pin
Title:
**30 pin, right angle connector**              <-- first line from input file
```

```
all lengths multiplied by 0.001 to convert to meters
Total number of filaments before multipole refine: 290
Total number of filaments after multipole refine: 455      <-- multipole needed
                                                               to refine (-a)
Multipole Summary
  Expansion order: 2                                   <--  -o option
  Number of partitioning levels: 3                     <--  -l option
  Total number of filaments: 455
Percentage of multiplies done by multipole: 100%
Scanning graph to find fundamental circuits...         <-- find meshes
Number of Groundplanes : 0                             <-- some data about
Number of filaments: 455                                   input geometry
Number of segments:  455
Number of nodes:     605
Number of meshes:    60
          ----from tree:                  60    <--fundamental circuits
          ----from planes: (before holes) 0                  from graph
Number of conductors:        30   (rows of matrix in Zc.mat)
Number of columns:           30   (columns of matrix in Zc.mat)
Number of real nodes:455
filling M...
filling R and L...
Total Memory allocated: 1985 kilobytes           <-- Total memory consumed so
Frequency = 10000                                      far (more consumed later)
Forming sparse matrix preconditioner.. <-- form preconditioner for this freq.
conductor 0 from node npin4_5_1        <-- do 1st column of admittance matrix
Calling gmres...
1 2 3 4 5 6 7 8                        <-- Iterations until convergence
conductor 1 from node npin4_4_1        <-- Begin next column
Calling gmres...
1 2 3 4 5 6 7 8
conductor 2 from node npin4_3_1
Calling gmres...
1 2 3 4 5 6 7 8
conductor 3 from node npin4_2_1
 .
 .
 .
conductor 28 from node npin0_1_1
Calling gmres...
1 2 3 4 5 6 7 8
conductor 29 from node npin0_0_1
Calling gmres...
1 2 3 4 5 6 7 8

All impedance matrices dumped to file Zc.mat    <-- Where the results are

Times:  Read geometry   1.51                    <-- some execution time info
```

```
        Multipole setup 11.94
        Scanning graph  0.01
        Form A M and Z  0.06
        form M'ZM       0
        Form precond    1.36
        GMRES time      75.92
   Total:               90.8
90.980u 0.640s 1:36.62 94.8% 321+2448k 0+0io 4pf+0w
prompt %
```

2.2 Processing the Output

2.2.1 The impedance matrix

The file `Zc.mat` is a text file containing the impedance matrices for the frequencies requested. The file `ReadOutput.c` in directory `src/misc/` is an example program for reading the text file for whatever processing is necessary. It contains the function `ReadZc()` which reads from a file and returns a linked list of impedance matrices and their corresponding frequencies. See the source file for more details. This function can be extracted and included in whatever program the user desires.

The function `main()` is provided as an example use of `ReadZc()`. For each of the matrices, it divides the imaginary part by the frequency to give the matrix $R + jL$ and then dumps the result to the standard output.

`ReadOutput.c` can be compiled by typing

```
cc -o ReadOutput ReadOutput.c
```

Here is a sample of its output after processing `Zc.mat` produced by running `fasthenry` on example file `onebargp.inp`:

```
prompt % ReadOutput Zc.mat
Not part of any matrix: Row 2 :  nodein  to  nodeout
Not part of any matrix: Row 1 :  nb1  to  nbout
Reading Frequency 10000
Reading Frequency 100000
Reading Frequency 1e+06
Reading Frequency 1e+07
Reading Frequency 1e+08
freq = 1e+08
Row 0: 0.00112838+3.50478e-08j -2.21062e-05-7.0003e-09j
Row 1: -2.23118e-05-6.99564e-09j 4.83217e-05+2.02958e-08j
freq = 1e+07
Row 0: 0.00112837+3.50478e-08j -2.21055e-05-7.0003e-09j
Row 1: -2.2311e-05-6.99564e-09j 4.83217e-05+2.02958e-08j
freq = 1e+06
Row 0: 0.0011272+3.50501e-08j -2.20335e-05-7.00045e-09j
Row 1: -2.22379e-05-6.99578e-09j 4.83168e-05+2.02958e-08j
freq = 100000
```

```
Row 0: 0.00106093+3.51782e-08j -1.7938e-05-7.00794e-09j
Row 1: -1.80765e-05-7.00341e-09j 4.80425e-05+2.02964e-08j
freq = 10000
Row 0: 0.000955377+3.58555e-08j -1.25489e-05-7.01913e-09j
Row 1: -1.25828e-05-7.01518e-09j 4.75474e-05+2.03045e-08j
prompt %
```

2.2.2 Creating Equivalent Circuits

Two approaches for generating spice equivalent circuits are available:

Approach 1: An equivalent circuit for a single frequency.

- Run fasthenry for EXACTLY one frequency.
- Compile (with some C compiler) the file MakeLcircuit.c in the directory fasthenry-3.0/src/misc/

```
cc -o MakeLcircuit MakeLcircuit.c -lm
```

- Run MakeLcircuit on the Zc.mat produced by FastHenry

```
MakeLcircuit Zc.mat > my_circuit.spice
```

This produces a circuit where nodes 0 and 1 correspond to the plus and minus nodes of the first FastHenry port (also called .external). 2 and 3 correspond to the second port. 4 and 5 the third, etc.

Approach 2: A circuit which models the frequency dependent resistances and inductances

This approach generates a reduced order model for the system using the arguments "-r n -M" to FastHenry. This gives an equivalent circuit which is valid for a range of frequencies.

- Run FastHenry: `fasthenry -r n -M myinput.inp`
  Where "n" is replaced with the desired order, say 20. This produces the file `equiv_circuitROM.spice` containing a single spice subcircuit.
- Include the subcircuit in a spice input file and connect devices to it.

The subcircuit will be named `ROMequiv` and its port nodes are p0 and m0 for the plus and minus terminals of port 0, p1 and m1 for port 1, etc. See the header of the equiv_circuitROM.spice file for more description. Any suffix specified with "-S" will be appended to both the subcircuit name, `ROMequiv`, and the subcircuit filename, `equiv_circuitROM.spice`.

Comments: Approach 1 will not model frequency dependent resistance and inductance since it gives an R and L at the single specified frequency. Approach 2 will model the full

frequency dependent effects up to some frequency where that frequency is higher for a higher chosen order, n. The reduced order model of Approach 2 is converted to a circuit from its state-space form through capacitors, resistors, and VCCSs so insight can only be gained by simulating the subcircuit in spice rather than looking inside the subcircuit file. An example is provided below. For a description of reduced order modeling for inductance computation see the paper `tchmt-epep94.ps` in the `pub/fasthenry` directory at the `rle-vlsi.mit.edu` ftp site:

> L. Miguel Silveira and Mattan Kamon and Jacob K. White, "Efficient Reduced-Order Modeling of Frequency-Dependent Coupling Inductances associated with 3-D Interconnect Structures", *IEEE Transactions on Components, Hybrids, and Manufacturing Technology, Part B: Advanced Packaging*, Vol. 19, No. 2, May 1996, pp. 283-288.

An example of Approach 2

The file `examples/pin-con7.inp` is 7 pins of a larger package shown in Figure 8. Each conductor is discretized into filaments to capture skin and proximity effects up to around 10^{10} Hz. One can generate an equivalent circuit via Approach 2 with

```
fasthenry -r 20 -M pin-con7.inp -S_pin_con7_r20
```

which produces the file `equiv_circuitROM_pin_con7_r20.spice` which is an equivalent circuit for a 140th $(20 * 7)$ order model. To test its accuracy, consider computing impedance matrices at many frequencies with

```
fasthenry pin-con7.inp -S_pin_con7
```

which produces `Zc_pin_con7_r20.mat` which is the impedance matrix for frequencies $1, 10, 10^2, \ldots, 10^{12}$. A spice file is included in the `examples` directory to test this subcircuit and put the results in `rom_check_con7_r20.out`:

```
spice3 -b -o rom_check_con7_r20.out  rom_check_con7_r20.ckt
```

The spice file in `rom_check_con7_r20.ckt` has a spice `.include` line to include the subcircuit of file `equiv_circuitROM_pin_con7_r20.spice`. It then does an AC analysis of the response of the circuit when 1 Amp is applied to port 0 and 0 Amp to the other ports. The results correspond to column 1 of the impedance matrix, Zc.

A comparison of the accuracy of entry (1,1) of the impedance matrix is given in Figure 12 for both "-r 7" and "-r 20".

As can be seen in the figures, in the order 7 reduced-order model the resistance matches the directly computed values (Zc.mat) up to about 10^8 Hz. The order 20 reduced-order model is more accurate, and represents the frequency-dependent resistance up to 10^9 Hz. The inductance in the example chosen does not have a strong frequency dependence.

Note that if Approach 1 were used, a single frequency would have to be chosen to generate an equivalent circuit. This would be less accurate but would probably reduce subsequent Spice simulation time.

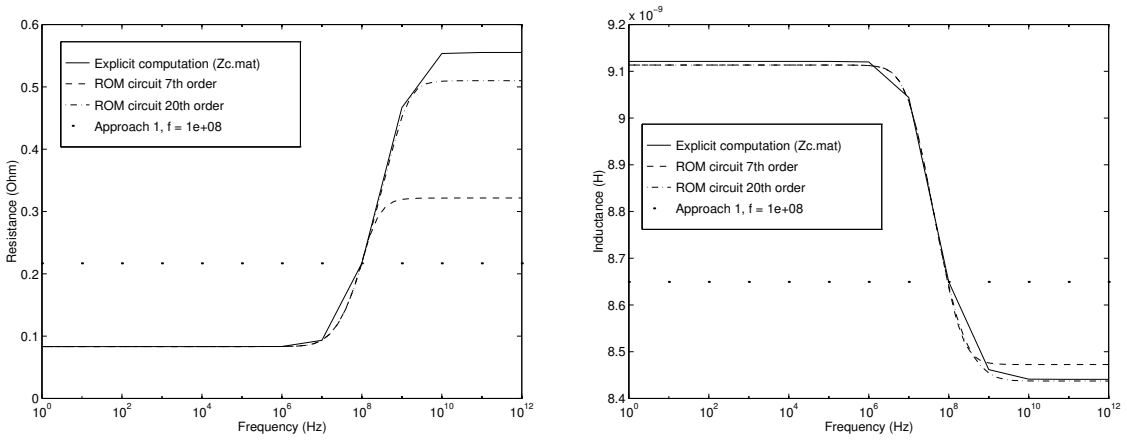

Figure 12: Comparison of impedance computation done via explicit solution at specified frequency points and reduced-order models

2.3 Command Line Options

This section describes using the command line options to change the defaults settings. All arguments are case insensitive.

`-` (dash) Forces input to be read from the standard input.

`-s` {`ludecomp` | `iterative`} - Specifies the matrix solution method used to solve the linear system arising from the discretization. `iterative` uses the GMRES iterative algorithm and `ludecomp` uses LU decomposition with back substitution. In general, GMRES is faster, however some speed up may be obtained using LU decomposition for problems with fewer than 1000 filaments. `iterative` is the default.

`-m` {`direct` | `multi`} - Specifies the method to use to perform the matrix-vector product for the iterative algorithm. `direct` forms the full matrix and performs the product directly. `multi` uses the multipole algorithm to approximate the matrix-vector product. For larger problems, the multipole algorithm can save both computation time and memory. `multi` is the default.

`-p` {`on` | `off` | `loc` | `posdef` | `cube` | `seg` | `diag` | `shells` } - Specifies the method to precondition the matrix to accelerate iteration convergence. Two classes of preconditioners are implemented in FastHenry. Local inversion preconditioners (`loc` and `posdef`) were used in prior releases and have been replaced by the sparsified-L preconditioners. Three types of sparsified-L preconditioners are available, (`cube`, `seg`, and `diag`). `cube` is the default and produces the best results, however under certain conditions can consume significantly more memory than `diag`. If there are multiple filaments in each segment, than `seg` is a middle ground between `diag` and `cube`, otherwise it is identical to `diag`. `shells` uses a current shell idea described in

> Mattan Kamon, Byron Krauter, Joel Phillips, Lawrence T. Pileggi, Jacob White, "Two Optimizations to Accelerated Method-of-Moments Algorithms

for Signal Integrity Analysis of Complicated 3-D Packages", *Proceedings of the IEEE 4th Topical Meeting on Electrical Performance of Electronic Packaging*, Portland, OR, October 1995, pp. 213-216.

The method is relatively new, and its performance is sensitive to the shell radius set by the `-R` option described below.

`-o n` - Specifies n as the order of multipole expansions. Default is 2. Choosing values less than 2 result in faster execution at the expense of poorer accuracy. Changes in the order of expansions should accompany changes in iteration error tolerance with the `-t` option.

`-l` {`n | auto`} - Specifies n as the number of partitioning levels for the multipole algorithm. `auto` chooses the level automatically and is the default.

`-f` {`off | simple | refined | both | hierarchy` } - Switches FastHenry to visualization mode only and specifies the type of FastCap generic file to make (for visualization ONLY!). `off` will produce no file and is the default. `simple` will produce a file named `zbuffile` from the segments defined in the input file. `refined` produces a similar file, named `zbuffile2`, but uses the segments after either user refinement specified with `-i` or required refinement necessary for accuracy of the multipole algorithm (`-l` option). `both` produces both files. One FastCap "panel" is created for each of the four sides along the length of the each segment. Reference planes are handled differently as described under the `-g` option below. See Section 3 for details on producing postscript. FastHenry will exit after creating the visualization files. `hierarchy` will dump a 2D representation of the nonuniform reference plane discretization hierarchy to the FIXED file `hier.qui` which can also be processed like the `zbuffile`.

`-g` {`on | off | thin | thick`} - controls appearance of the ground plane when using the `-f` option. `off` is the default and only four panels are produced for each plane, one for each of the edges. Thus only the outline of each plane is drawn. This makes generation of the postscript image much faster and also makes the planes transparent. No holes are visible, however. `on` or `thin` will draw all the overlapping segments of the ground plane as if infintely thin. Note that using this option for a finely discretized plane may take a long time to generate a postscript image with `zbuf`. `thick` will completely draw the segments of the plane and will be even slower for generating postscript images.

`-a` {`on | off`} - `on` allows the multipole algorithm to automatically refine the structure as is necessary to maintain accuracy in the approximation. The structure will be refined whether or not the multipole algorithm is used. `off` prevents refinement and will produce a warning if the multipole algorithm is used and prevented from necessary refinement. `on` is the default and is equivalent to `-l auto`. Note that this is not refinement to reduce discretization error. See the `-i` option.

`-i n` - Specifies n as the level for initial refinement. This option allows the user to refine the structure if the input file is too coarse. It will divide each segment of the geometry into multiple segments so that no segment has a length greater than $\frac{1}{2^n}$ times the length of the smallest cube which contains the whole structure. The default is 0 (no refinement). This option is rarely used since the multipole algorithm will refine as it needs and discretization errors of this sort are usually small.

`-d` {`on` | `off` | `mrl` | `mzmt` | `grids` | `meshes` | `pre` | `a` | `m` | `rl` | `ls`} - dump certain internal matrices to files. The format of some of the files can be specified with the `-k` option. `on` dumps the M, R, L, MZM^t and preconditioner matrices. `off` dumps none and is the default. `mrl` dumps the M, R, and L matrices. `mzmt` dumps the MZM^t matrix for $w = 1$. `grids` dumps matrices for viewing the current distribution inside each reference plane (only in matlab format). `meshes` is a matlab file for viewing the KVL meshes chosen by FastHenry. `pre` dumps the preconditioner in a sparse matrix text format. `ls` dumps the sparsified L matrix in sparse text format. `a` is the branch incidence matrix, A. `m` can be used to dump only the mesh matrix M. Similarly, only R and L can be dumped with `rl`. The right-hand-side vector, V_s, is dumped to the file `b.mat` whenever anything but `off` is specified.

`-k` {`matlab` | `text` | `both`} - Specifies type of file to dump with the `-d` option. `matlab` dumps the files as M.mat, MRL.mat, MZMt.mat, and A.mat in a format readable by matlab. `text` saves the files M.dat, L.dat, R.dat, MZMt.dat, and A.dat as text. `both` saves files in both formats.

`-t rtol, -b atol` - Specifies the tolerance for iteration error. FastHenry calculates each column of the impedance matrix separately. The iterative algorithm will stop iterating when both the real and imaginary part of each element, x_k, of the current column being calculated satisfy

$$|x_k^{i-1} - x_k^i| < rtol * (|x_k^i| + atol * (max_j |x_j^i|)) \tag{1}$$

where i is the iteration number. The defaults are $rtol = 10^{-3}$ and $atol = 10^{-2}$. In essence, this causes GMRES iterations to stop if every element of solution vector has changed by less than the fraction $rtol$ since the last iteration. If, however, element x_k is smaller than $atol$ times the largest element in the solution vector, then the stopping criteria is less stringent. Note that real and imaginary parts are treated separately.

`-c n` - n = maximum number of iterations to perform for each solve (column). Overrides the default of 200.

`-D` {`on` | `off`} - Controls the printing of debugging information. `off` is the default. `on` will cause FastHenry to print more detailed information about the automatic partitioning level selection, memory consumption, preconditioner calculation, and convergence of the iterates. The Matlab file `Ycond.mat` is also produced which contains the admittance matrices at each frequency and also the norm of the residual at each step of the GMRES algorithm.

`-x portname` - Specifies that only the column in the admittance matrix specified by `portname` should be computed. Multiple `-x` specifications can be used. In this case, the output file `Zc.mat` will contain the requested columns of the admittance matrix instead of the impedance matrix. The portname is specified in the input file with the `.External` keyword. The primary uses of this option are either to exploit symmetry or to compute single columns for observing current distribution. For instance, if a 10 pin package is symmetric about some axis, then only five columns need be computed. The user is then responsible for forming the 10×10 impedance matrix from the 10×5 result. Also, if the `-d grids` option is used, the port which is the source of current would be specified with this option.

`-S suffix` - This adds the string `suffix` to all filenames for this run. For instance, `-S _blah` will produce the output file `Zc_blah.mat`.

`-r order` - Specifies a reduced order model of the system as output. The size of the model will be `order`*number-of-ports. A matlab file `rom.m` will contain the A,B,C,D matrices representing the state-space reduced order model. More useful is the file `equiv_circuitROM.spice` which is a spice3 compatible subcircuit which can be included for circuit simulation.

`-M` - If `-r` is specified with a nonzero order, then this option will cause FastHenry to exit after generating a reduced order model.

`-R radius` - If `-p shells` is specified, then this specifies the radius of the shells to use.

`-v` - Regurgitate internal representation of the geometry to stdout in the input file format. Good for debugging. User can compare output to the original input file. Also, by altering the file `regurgitate.c`, the user can also call functions to translate and reflect the geometry for use as another input file.

2.4 Discretization Error Analysis

This section gives some guidelines and describes an experimental approach to determining an appropriate discretization to model skin and proximity effects in long, thin condutors defined with FastHenry segments. (For a description of discretization for reference planes, see "nonuniform_manual.ps".)

To begin, assume it is desired to compute the impedance matrix for the pin connect structure of Figure 8 for frequencies up to 10^8 Hz.

2.4.1 The DC case

If only the DC case were of desired, it could be computed rapidly since there is no skin effect. The number of filaments per segment for the entire structure is set to one by setting `nhinc=1` and `nwinc=1` with the line

```
.DEFAULT Z=85. H=8.5 W=24. nhinc=1 nwinc=1
```

In order to extract a nonzero value for the inductance, the frequency is set small, but nonzero

```
.freq fmin=1e0 fmax=1e0 ndec=1
```

Since there are only a few hundred filaments, there is no advantage to the multipole algorithm so LU decomposition can be used to save a few seconds:

```
fasthenry pin-connect.inp -sludecomp -S _DC
```

The output will be in file `Zc_DC.mat`

2.4.2 The highest frequency case

Next, consider deciding how many filaments are needed to accurately compute the resistance and inductance at the highest frequency point of interest, $f = 10^8$ Hz. A good rule of thumb is to choose the discretization such that the width of the smallest filament is roughly equal to the skin depth. For this geometry, $\rho = 0.0238$ ohm-mil or $\sigma = 1.641 \times 10^6$ mho/m giving a skin depth of

$$\delta = \sqrt{\frac{1}{\pi f \mu \sigma}} = 1.55 \text{ mils.} \tag{2}$$

To observe the error involved with various discretizations, we will look at the impedance matrix for two typical adjacent segments from the `pin-connect.inp` example:

```
***Two segments from:  PACKAGE INDUCTANCE - 35 of 68 pins***
*    For experimenting with skin/proximity effect modeling
*
.UNITS MILS
.DEFAULT RHO=.0238
.DEFAULT Z=85. H=8.5 W=24. nhinc=3 nwinc=5 rh=2 rw=2
*
* BEAM 16
*
N16B Y=100. X=387.5
N16C Y=100. X=493.
E16C N16B N16C
.EXTERNAL N16B N16C port16
*
* BEAM 17
*
N17B Y=50. X=387.5
N17C Y=50. X=493.
E17C N17B N17C
```

```
.EXTERNAL N17B N17C port17
*
.freq fmin=1e1 fmax=1e12 ndec=3

.END
```

The result will be a 2×2 complex matrix for each frequency point,

$$\mathbf{Z}(\omega) = \begin{bmatrix} R_{11}(\omega) + j\omega L_{11}(\omega) & R_{12}(\omega) + j\omega L_{12}(\omega) \\ R_{21}(\omega) + j\omega L_{21}(\omega) & R_{22}(\omega) + j\omega L_{22}(\omega) \end{bmatrix} \tag{3}$$

If we choose $nhinc = 3$ and $nwinc = 5$ with a default ratio of adjacent filaments equal to 2 ($rh = rw = 2$), then the width of the smallest filament on the right and left of the segment will be 1/10 the width $((1+2+4+2+1)^{-1})$, or 2.4 mils. The thickness of the filaments on the top and bottom of the segment will be 1/4 of the height $((1+2+1)^{-1})$ or 2.1 mils.

To run fasthenry,

```
fasthenry pin-con2seg.inp -sludecomp -aoff -S _3x5_2
```

In this case, since there are so few filaments, LU decomposition is slightly faster. Also, automatic refinement is prevented with the `-aoff` option since the multipole algorithm is not being used and this problem is essentially two-dimensional.

The smallest filaments are slightly above the skin depth and thus the results are inaccurate at $f = 10^8$ as shown by the dotted lines in Figure 13. The next logical step might be to increase the discretization to $nhinc \times nwinc = 4 \times 6$ however using even numbers of filaments is not efficient since the center of the segment, where current is relatively constant, will be divided into 4 separate filaments. For the odd case, the center is modeled by only one filament. Instead, consider changing the ratio of adjacent segments to $rh = 4$, $rw = 3$. Now, the width of the smallest filament on the left and right sides of the segment will be 1/17 of the total width $((1+3+9+3+1)^{-1})$ or 1.41 mils and the thickness of the smallest filament on the top and bottom, 1.42 mils. This is much closer to the skin depth and gives better results as shown by the dash-dot lines in Figure 13.

Another possibility is to reduce the number of filaments below 3×5 but further increase the ratio of adjacent filaments. This change would result in faster execution since there would be fewer filaments, however some accuracy at midrange frequencies would be sacrificed. For instance, consider maintaining the width of the smallest filaments by changing the discretization above to 3×3 with $rh = 4$, $rw = 15$. In this case there are too few filaments to model the decay of current density from the outside edge to center for $f \in [10^6, 10^8]$ as shown by the dashed line in Figure 13.

After the user has decided which error for the above cases is tolerable for a given problem, the discretization can then be used for the full `pin-connect.inp` example. The default $nhinc$ and $nwinc$ must be set with

```
.DEFAULT Z=85. H=8.5 W=24. nhinc=3 nwinc=5 rh=4 rw=3
```

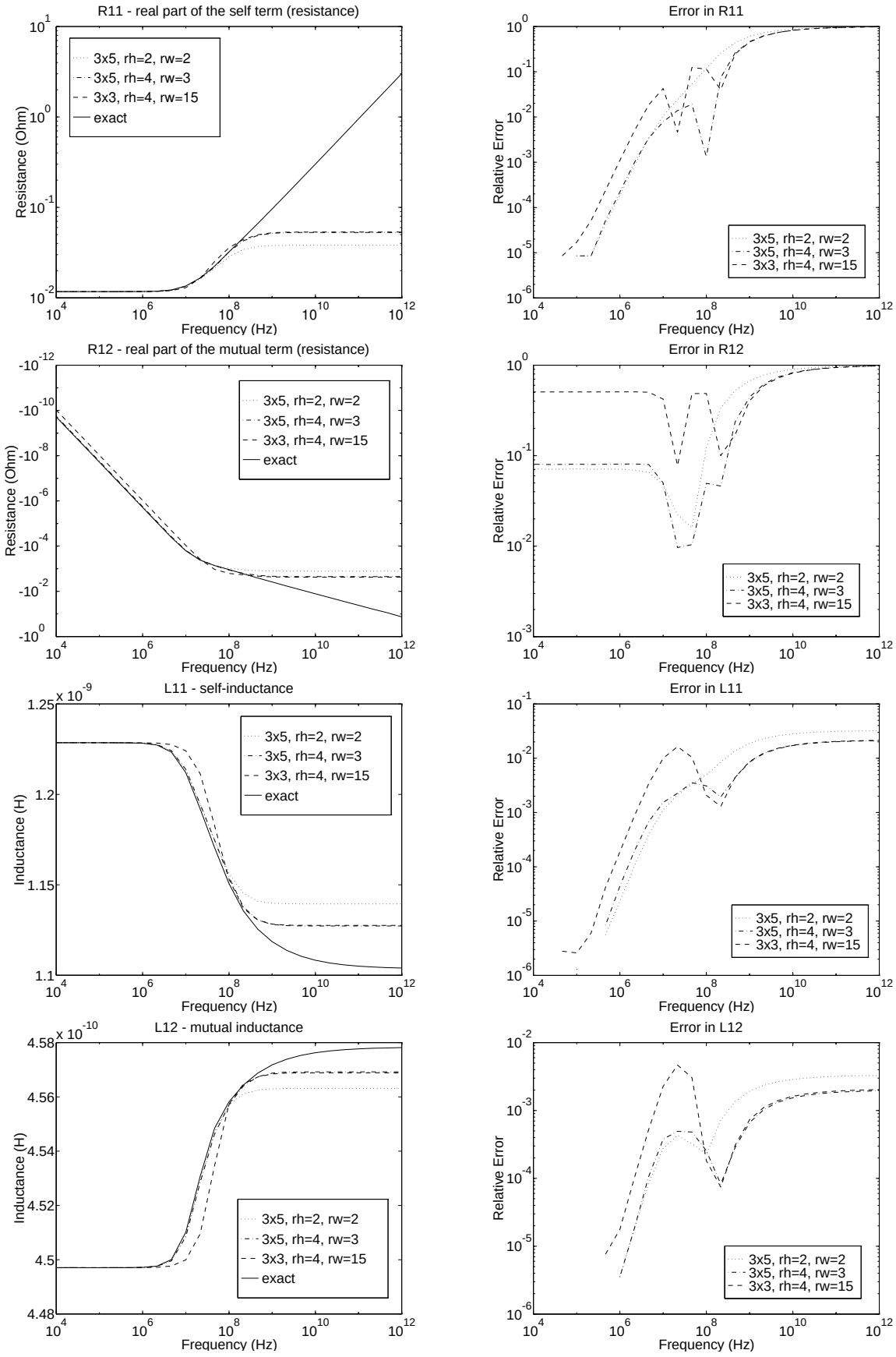

Figure 13: Values and errors for the impedance matrix as a function of frequency for the 2 segment experiment

and the frequencies of interest with

```
.freq fmin=1e0 fmax=1e12 ndec=3
```

The frequency setting above was used to generate Figure 13. Note that with the sparsified-L preconditioner, used by default, the low frequency impedance matrix is computed in very few iterations. Thus, while the low frequency case could have been computed with a coarser discretization as in Section 2.4.1, there is not much added execution time in computing it at a finer discretization.

The exact values in Figure 13 were generated using a 21×25 discretization with $rh = rw = 2$. It was run with

```
fasthenry pin-con2seg.inp -aoff -mdirect
```

The multipole algorithm would require space be divided, and thus the filaments also be divided, tripling the number of filaments in this case. Since the original number of filaments is small and, again, this is essentially a two-dimensional problem, we can avoid both dividing space (`-aoff`) and using the multipole algorithm (`-mdirect`). The problem is small enough to fit in memory with all the matrices formed directly however is large enough to warrant using the iterative algorithm instead of LU decomposition.

For the 21×25 problem, the time spent forming the preconditioner is significant. The `-pdiag` option could have been used to reduce the preconditioner time, however this would come at the expense of slower convergence especially at the higher frequencies.

3 Geometry Postscript Pictures

The ability to see the geometry under analysis is an important tool for debugging input files. This section describes how to generate postscript files for visualizing the three dimensional structures defined by a given input file. The process involves first running FastHenry to generate a file of "panels" and then running `zbuf` to generate the postscript file.

3.1 Creating the panel file

When FastHenry is given the `-f` option, it changes from calculation to visualization mode. It will produce a file of panels readable by the `zbuf` program. Each panel is a quadrilateral representing one of the faces of a segment in the input geometry. This file of panels can either be formed from the segments of the geometry before FastHenry's refinement with the `simple` argument, or after refinement with the `refined` argument. The basenames for the files produced are `zbuffile` for simple geometry and `zbuffile2` for refined unless the `-S` is used.

In addition, the file `zbuffile_shadings` or `zbuffile2_shadings` is produced which specifies a shade of gray to assign to each of the panels. `zbuffile_shadings` is produced for `-f simple` and shades each of the reference planes (if drawn with `-g on`) of the structure differently, and leaves all other structures white. `zbuffile2_shadings`, produced with `-f refined`, does the same, however it attempts to shade differently each of the conductors specified with a `.external` statement.

3.2 Creating the postscript file

The panel file described in the previous section can be rendered in postsciprt with the `zbuf` program. This program is a modified version of the algorithms used for visualization in the capacitance extraction program, FastCap, developed by Keith Nabors.

The complete command line format of the `zbuf` program is

```
zbuf [-a<azimuth>] [-e<elevation>] [<input-file>]
        [-r<rotation>] [-h<distance>] [-s<scale>] [-w<linewidth>]
        [-u<upaxis>] [-q] [-x<axeslength>]
        [-b<.figfile>] [-c] [-v] [-n] [-f] [-g] [-m]
```

Table 2 itemizes the options and Table 1 gives several examples of their use.

Usually the default settings produce an acceptable plot, but many adjustments are possible, the most important being the view angles (relative to a coordinate system parallel to the input coordinates and centered on the center of the object) which are adjusted using the `-a` and `-e` options. Other options control view distance (`-h`), two-dimensional projection scale (`-s`), rotation (`-r` and `-u`) and line width (`-w`). Axes of any length may be included with the `-x` option and lines, arrows and dots may be added using `-b`. Shading can be accomplished with the `-q` option. Table 1 gives the commands used to produce the line drawings in this guide as examples.

In version 3.0, the `zbuf` program now takes the "-m" argument to produce a Matlab file for faster visualization in matlab. This is very beneficial for large files since producing the postscript file with `zbuf` can take n^2 time. The matlab file can be viewed within matlab with the fasthenry-3.0/bin/plotfastH.m matlab function. The file `zbuffile.mat` would be produced with "zbuf -m zbuffile" which can then be viewed in matlab with "plotfastH('zbuffile.mat')". Also, you can modify the file `src/zbuf/dump_struct.c` to output in YOUR own format instead of matlab.

3.3 Tricks

For Figure 7, the shading of the trace and ground plane were swapped by negating all the shading values with the `awk` file `invert.awk` provided in the `zbuf` source directory. Also, Figure 10, since it is only one pin, would normally appear white and all panels would have shade 0 in the shading file. By randomly changing one panel to -5 and another to 10, then -5 becomes white, 10 becomes black, and 0, a shade of gray. The single black panel is seen on the left end of the figure and the white panel is obscured by other panels.

3.4 Visualization artifacts

This section discribes possible problems with producing pictures. Many of the listed items come directly from the FastCap documentation.

1. To minimize the number of panels that `zbuf` must deal with, FastHenry does not output panels for the end faces of a segment. Thus, segments often appear hollow. Uncomment the appropriate regions in `writefastcap.c` to change this.

Figure	FastHenry/Zbuf Usage
5	`fasthenry gpexamp_copper.inp -f simple -S _pcb` `zbuf zbuffile_pcb`
6	`fasthenry vias.inp -f refined -g on` `zbuf -a60 -e60 zbuffile2 -q`

7	`fasthenry holey_gp.inp -f simple -g on`
	`mv zbuffile holey_gp`
	`awk -f src/zbuf/invert.awk zbuffile_shadings > holey_gp_shadings`
	`zbuf holey_gp -e0 -q -uy`

8	`fasthenry pin-connect.inp -f simple -S pin` `zbuf zbuffilepin -a25`
9	`fasthenry 30pin -f refined` `zbuf zbuffile -a30 -q`
10	`fasthenry one_of_30pin -f simple` `zbuf zbuffile -q -e20 -a0`

Table 1: Commands used to generate some representative figures in this guide.

`zbuf` Options

Option	Default	Range	Function
`-a`	`50.0`	†	Specifies azimuth view angle in degrees.
`-e`	`50.0`	†	Specifies elevation view angle in degrees.
`-r`	`0.0`	†	Specifies final rotation of 2-D image in degrees.
`-h`	`2.0`	$\geq$ `0.0`	Specifies distance from surface of object in object radii.
`-s`	`1.0`	$>$ `0.0`	Specifies final scaling of 2-D image.
`-w`	`1.0`	$\geq$ `0.0`	Specifies postscript file line width.
`-u`	`z`	`x`, `y`, or `z`	Specifies which 3-D axis is mapped to y-axis in 2-D image.
`-q`	—	—	Uses zbuffile_shading shadings file to give grayscale shading to conductors.
`-x`	`1.0`	$>$ `0.0`	Includes axes of length `axeslength` in picture.
`-b`	◇	◇	Specifies lines, dots and arrows to superimpose on picture.
`-c`	—	—	Puts the command line in postscript file pictures.
`-v`	—	—	Removes `showpage` from postscript file.
`-n`	—	—	Numbers faces in order input.
`-f`	—	—	Suppresses hidden line removal.
`-g`	—	—	Prints the graph used to order the panels in postscript file.
`-m`	—	—	Produces a matlab file rather than a postscript file

Table 2: Zbuf Options

†Range is unrestricted.
◇ See function `readLines()` in `src/zbuf/zbufInOut.c` for a description of the `.fig` file format.

2. `zbuf` sometimes produces panel ordering errors (and usually print warning messages) when panels intersect. However, often the postscript file is correct.

3. Occasionally the postscript pictures are incorrect even for legal discretizations. There are two known bugs leading to this problem. The first is often avoided by changing the view angle slightly. The second is caused by problems with dimensions on the order of 10^{-4} or less. This problem can only be avoided by rescaling the input by changing the `.Units` to `km`.

4. The panel sorting algorithm used to write out the postscript files takes time proportional to the number of panels squared. This limits its usefulness to problems with fewer than a few thousand panels.

5. The shading algorithm uses one shade of gray to shade all the segments along only one path between the two nodes of a `.external` statement. Thus, if there are multiple paths, such as in Figure 9, the other paths will appear white.

6. When axes are included in postscript files using the `-x<axes length>` option, the axes' two-dimensional projections appear in the postscript file before the panel fills. This means that the object should be between the view point and the axes, otherwise the axes can be obscured strangely. Thus the option works best when viewing objects that lie entirely in the positive orthant from a view point in the positive orthant.

4 Reference Plane Current Visualization

When FastHenry is given the `-d grids` option, it dumps Matlab readable files of the current distribution in each reference plane of the geometry under analysis. One file is dumped for each of the columns computed for the admittance matrix. The file holds the current distribution produced by setting the source across the port corresponding to the current column to one volt and all others ports to zero. This process is then repeated for each freqeuncy point.

Note: For nonuniformly discretized planes, the filename is the same, but the file format is very different. See `nonuniform_manual.ps`. This document also describes viewing the current distribution in other than reference planes.

4.1 Current Files

The files are named according to the column in `Zc.mat` and frequency point. They have the form Grid$n$_m.mat where n is the column (and row) in `Zc.mat` of the port which has the one volt source and m is the frequency point number. The frequency points start at 0 and increment by one for each new frequency.

Inside each file are matrices for each reference plane named `grid1g<name>` and `grid2g<name>` where *name* is the name of the reference plane defined in the input file. Each entry in `grid1g<name>` is the complex valued current phasor for a filament in the `seg1` direction. `grid2g<name>` corresponds to filament currents in the `seg2` direction.

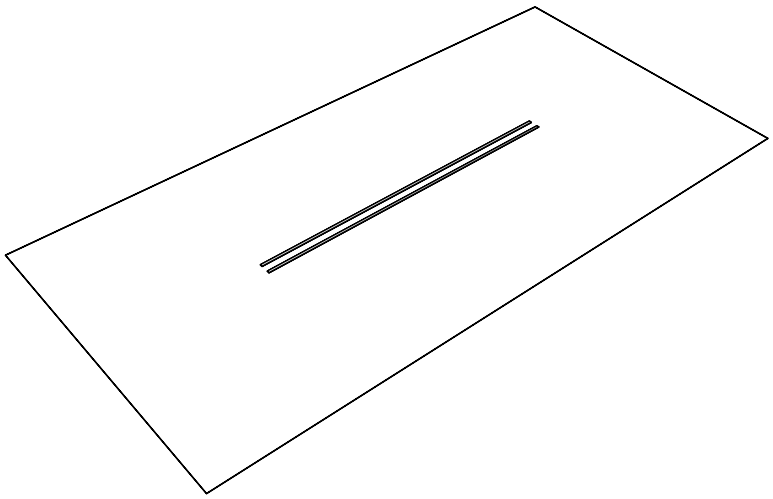

Figure 14: Two traces over a ground plane

In general it is difficult to visualize complex current, but when the current is purely real (DC) or purely imaginary (high frequency), these two matrices can be thought of as the current in two orthogonal directions.

4.2 Examples

4.2.1 Traces over a solid plane

Consider the example file `together.inp` shown in Figure 14. This structure is simply two traces passing over a ground plane with one end of each trace shorted to the plane. Thus, when one volt is applied to the other end of one of the traces, current travels down the trace and returns through the plane. To generate the current distribution files:

```
fasthenry together.inp -d grids -x trace1
```

The files `Grid1_0.mat` and `Grid1_1.mat` are produced in addition to FastHenry's normal output files. By default, FastHenry would compute the current distribution for the source voltage placed one trace and then again for the other trace. Since the current distribution is nearly identical for both cases, we need only compute one of these cases and thus the `-x` option is used to specify that only the column of the admittance matrix corresponding to port `trace1` is to be computed (see input file). Also, since visualization is the goal, `together.inp` specifies that only a low frequency case, $f = 10^{-1}Hz$, and a high frequency case, $f = 10^{19}Hz$ are to be computed. In this example, the discretization of the plane is coarse for visualization purposes.

The current distribution can then be viewed using the following Matlab commands to produce Figures 15 and 16.

```
< M A T L A B (R) >
(c) Copyright 1984-93 The MathWorks, Inc.
All Rights Reserved
```

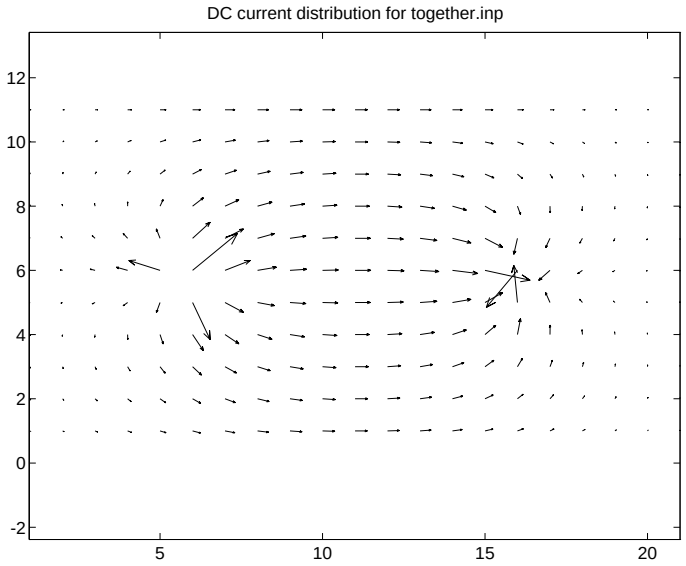

Figure 15:

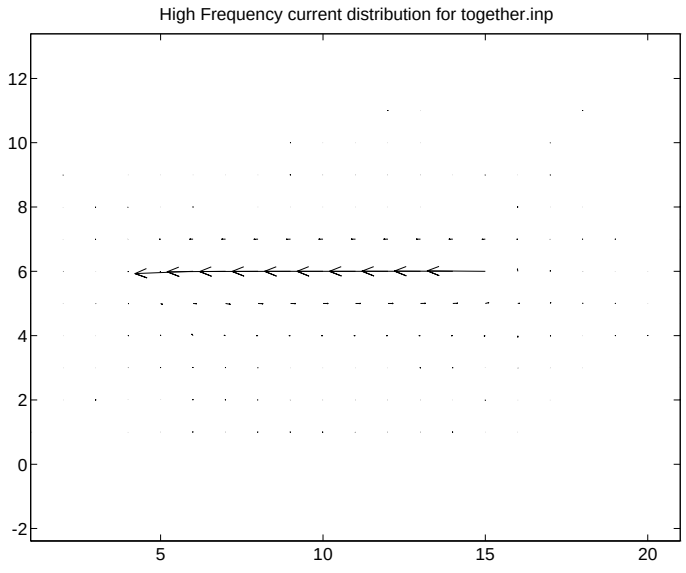

Figure 16:

```
                                Version 4.1
                                Jun 15 1993

Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info, subscribe

>> load Grid1_0.mat
>> who

Your variables are:

grid1g1   grid2g1

>> quiver(real(grid1g1),real(grid2g1),2)
>> title('DC current distribution for together.inp')
>> print together_DC.ps
>> load Grid1_1.mat
>> who

Your variables are:

grid1g1   grid2g1

>> quiver(imag(grid1g1),imag(grid2g1),2)
>> title('High Frequency current distribution for together.inp')
>> print together_highf.ps
```

Note that for the DC case in Figure 15 the current spreads across the plane as it travels from its source to sink points but at high frequency the current is focused underneath the trace. Note also that the high frequency current travels in the opposite direction as the current at DC since $I = V/(jwL) = -jV/(wL)$.

4.2.2 Traces over a divided plane

Next consider the example file `broken.inp` shown in Figure 17. This is identical to `together.inp` except that the plane is now broken in two pieces and connected by copper "tethers" as shown.

After running FastHenry with

```
fasthenry broken.inp -d grids -x trace1
```

the two matrices can be concatenated in matlab to produce Figures 18 and 19 with the following commands:

```
>> load Grid1_0
>> who
```

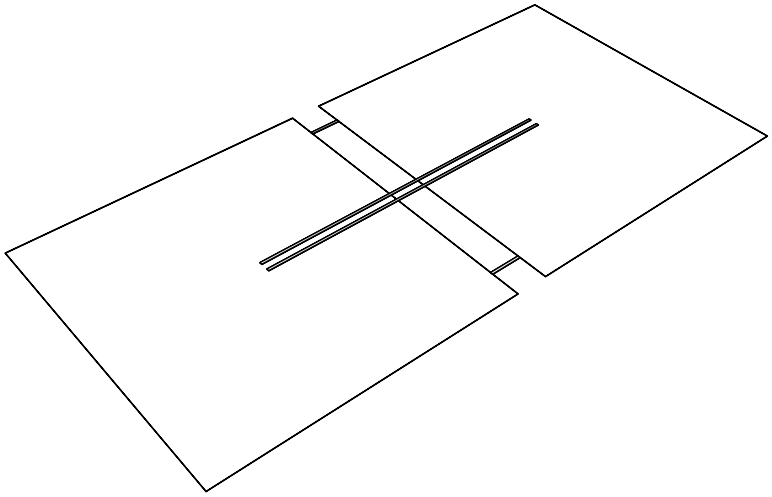

Figure 17: Two traces over a divided ground plane

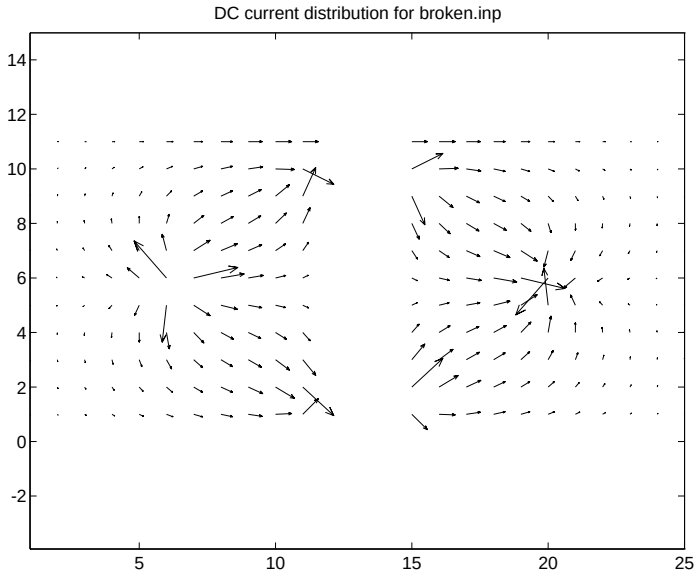

Figure 18:

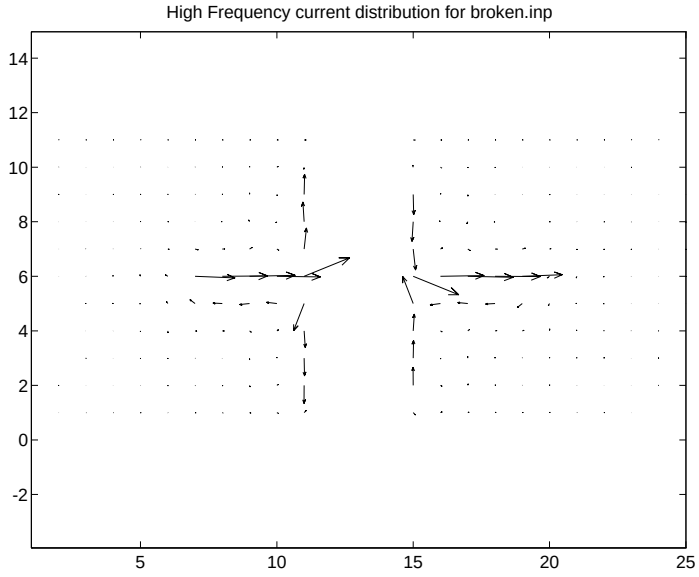

Figure 19:

```
Your variables are:

grid1g_one     grid2g_one
grid1g_two     grid2g_two

>> size(grid1g_one)

ans =

    11    11

>> space = zeros(11,3);
>> new1 = [-grid1g_two(:,11:-1:1) space grid1g_one];
>> new2 = [grid2g_two(:,11:-1:1) space grid2g_one];
>> quiver(real(new1),real(new2),2);

(similarly for Grid1_1.mat)
```

Note that some difficulty is involved in combining the matrices because the `seg1` direction in space is different for the two planes since the corners of the plane are defined in counter-clockwise order for one plane, and clockwise for the other.

From Figure 18 it may seem unusual that current can point into the empty space between the planes. This, however, is only an artifact of the method of visualization since the current vectors do not precisely represent the current flow at a particular point. Since the ground plane is discretized as a grid of segments (see Figure 4), one could attempt to represent the current as that passing through each node. But each node

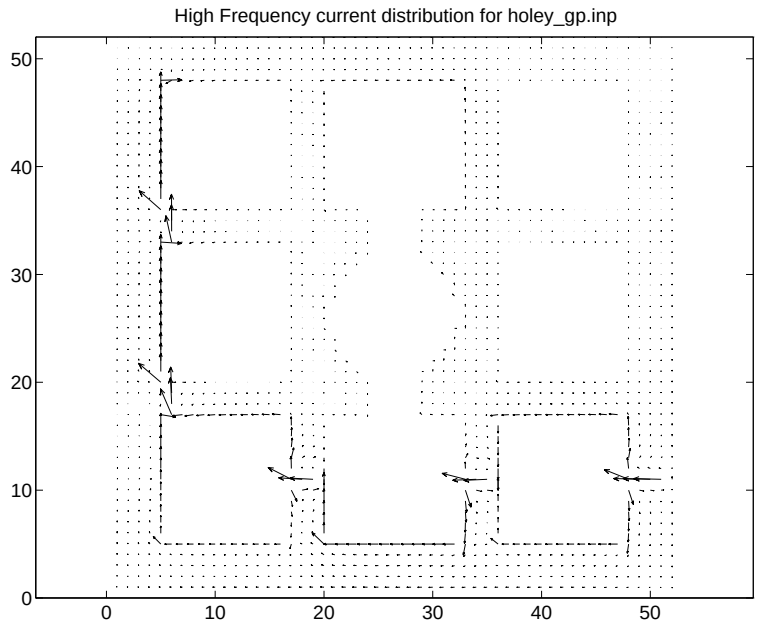

Figure 20:

on the plane has four segments attached to it while only two are needed to define a vector direction. FastHenry consistently picks the segments with the smaller indices. This choice can be altered in the function `makegrids()` in `fillM.c` if the user wishes to average the currents, for instance.

4.2.3 Trace over a plane with holes

As a final, more interesting example, consider observing the current for example file `holey_gp.inp` shown previously in Figure 7. To produce Figure 20,

```
fasthenry holey_gp.inp -d grids
```

and then in Matlab,

```
>> load Grid1_1
>> quiver(imag(grid1g1),imag(grid2g1),3)
```

Again in this high frequency case, the current is observed to try its best to stay underneath the trace, outlining the holes it must circumvent.

A Compiling FastHenry

A tar file containing the source files for `fasthenry` and this guide may be obtained on tape by sending a written request to

Prof. Jacob White
Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science
Room 36-880
Cambridge, MA 02139 U.S.A.

This address may also be used for general correspondence regarding `fasthenry`, although electronic mail may be sent to `fasthenry-bug@rle-vlsi.mit.edu`, for bug reports, and to `fasthenry@rle-vlsi.mit.edu`, for questions or comments, if it is more convenient. Comments on FastHenry are encouraged as well as comments on unclear portions of this manual.

The tar file has the form

```
fasthenry-3.0-9Oct96.tar.Z
```

and yields a one level directory when untarred with the commands

```
uncompress fasthenry-3.0-9Oct96.tar.Z
tar xvf fasthenry-3.0-9Oct96.tar
```

It will create a directory tree underneath `fasthenry-3.0` which contains all the C source files underneath `src/`, this manual in `doc/`, and the example files in `examples/`.

The tar files may also be obtained via anonymous ftp to `rle-vlsi.mit.edu`. Use username `anonymous` with your email address as the password. The FastHenry tar files are contained in directory `pub/fasthenry`. This directory also contains various publications on multiplole accelerated inductance extraction. The most complete description is the compressed postscript file `ms_thesis.ps.Z`:

> Mattan Kamon, *Efficient Techniques for Inductance Extraction of Complex 3-D Geometries*, M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA., February 1994.

`mtt.ps.Z` is a preprint of

> M. Kamon, M.J. Tsuk, and J. White, "FASTHENRY: A Multipole-Accelerated 3-D Inductance Extraction Program", *IEEE Transactions on Microwave Theory and Techniques*, Vol. 42, September 1994.

See the `README` file at the ftp site for update information on releases, manuals, publications.

A.1 Compilation Procedure

FastHenry is compiled by changing to the `fasthenry-3.0` directory, and typing

```
make all
```

to create the executables `fasthenry` and `zbuf` in the `bin/` directory.

If you are compiling on a DEC 5000, DEC 3000 (Alpha), SGI, or a system running System V (HP and Suns running Solaris, perhaps), other flags are required for compilation. In this case, before compilation as instructed above, give the command:

```
config <name>
```

where `<name>` is one of `dec`, `alpha`, or `sgi`, or `solaris` for compilation on a DEC 5000, DEC 3000 (Alpha), and Silicon Graphics workstation, respectively. Additional steps for compiling on an SGI are given in the file `README.sgi` in the `fasthenry-3.0` directory. `config sysV` is also provided for compilation on HP-UX, and other machines running System V, but has not been thoroughly tested. For more details on compilation, see the `fasthenry-3.0/README` file.

A.2 Producing this Guide

This guide is available in postscript as three separate files: `manual_001.ps, manual_002.ps`, and `manual_003.ps` in the `doc/` directory. `manual_001.ps` contains up through page 13, `manual_002.ps` is pages 14–31, and `manual_003.ps` is pages 32–36. Note that these files contain many detailed postscript images and may take significant time to print. The LaTeX version of the manual without the figures is also available.

Also, the nonuniform plane discretization manual is in the files `nonuniform_manual_1.ps` and `nonuniform_manual_2.ps`

B Changes in Version 3.0

- Specify a nonuniform discretization of a reference plane to capture small features in fewer elements.

- Two approaches for generating spice equivalent circuits are available:

  1. An equivalent circuit for a single frequency.

  2. A circuit which models the frequency dependent resistances and inductances through a reduced state-space representation

  Comments: Method 1 will not model frequency dependent resistance and inductance since it gives and R and L at the single specified frequency. Method 2 will model the full effects up to some frequency.

- Major Bug fix: For reference planes which form segments with different widths in the x-direction versus the y-direction, the sizes weren't computed correctly in version 2.0 and later. This has been fixed.

- The `zbuf` program now takes the "-m" argument to produce a Matlab file for faster visualization in matlab. This is very beneficial for large files since producing the postscript file can take n^2 time. The matlab file can be viewed within matlab with the fasthenry-3.0/bin/plotfastH.m matlab function. The file `zbuffile.mat` would be produced with "zbuf -m zbuffile" which can then be viewed in matlab with "`>> plotfastH('zbuffile.mat')`". Also, you can modify the file `src/zbuf/dump_struct.c` to output in YOUR own format instead of matlab.

- Sparse preconditioner. Specify `-p shells` to use a preconditioner based on current shell sparsification.
- Regurgitate the input file with `-v` to see what FastHenry thinks it has read. Also can translate and reflect geometry before output.