Nonuniformly Discretized Reference Planes in FastHenry 3.0

Mattan Kamon

10 October 1996

This is a version of an internal memo slightly modified for the FastHenry release. Nonuniformly discretized planes permit small features to be resolved without excessive numbers of elements. However these routines do not support the "hole" utility and have a different implementation for meshed planes. Part 1 describes the internal representation of nonuniform planes. Part 2 describes creating a discretization and Part 3 includes examples.

Summary:

This document describes the implementation of a scheme for nonuniform discretizations of planar elements in the 3D inductance extraction code FastHenry[Kamon]. FastHenry uses the Partial Element Equivalent Circuit (PEEC) approach [Ruehli] to compute frequency dependent resistance and inductance of a 3D geometry of conductors. The PEEC approach is straightforward and very efficient for long thin conductors for which the current can be assumed to flow in one vector direction. For conductors such as ground planes or substrates, which have inherently 2D or even 3D distributions, the PEEC approach still can be used but care must be taken in the discretization of the plane to insure greatest efficiency.

Version 2.0 of FastHenry can model 2D structures, however only allows the uniform discretization of such structures. This greatly restricts efficiency since features which require a fine discretization locally such as contacts and traces, force a fine discretization over the entire plane. The efforts of this project have been divided into two parts:

1. Enhancing FastHenry to use nonuniform discretizations.
2. Creation of appropriate discretizations

The enhancement of FastHenry has been the primary focus. The generation of discretizations have been given less attention since it can be done with minimal alterations to the core algorithms of FastHenry. This document gives the implementation details as well as a user's guide for the new features of FastHenry. The user's guide aspect is meant as a supplement to the original FastHenry version 2.0 user's guide.

This document is divided into 3 parts. The first two correspond to the two parts of the project described above. The third part gives numerous examples of using the code.

Part 1: Enhancements to Fasthenry

This section describes the modifications to FastHenry.

Specification of a nonuniform discretization

Given a uniform discretization as shown below, we wish to provide local refinement for contacts and regions below traces.

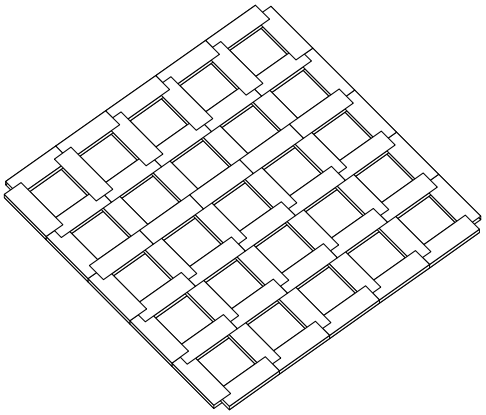


Figure: Uniform discretization of plane. Segments one third actual width for illustration

To accomplish this goal, let each segment of conductors be a line segment, and let the region bounded by four line segments be a “cell” as shown in Figure 2.

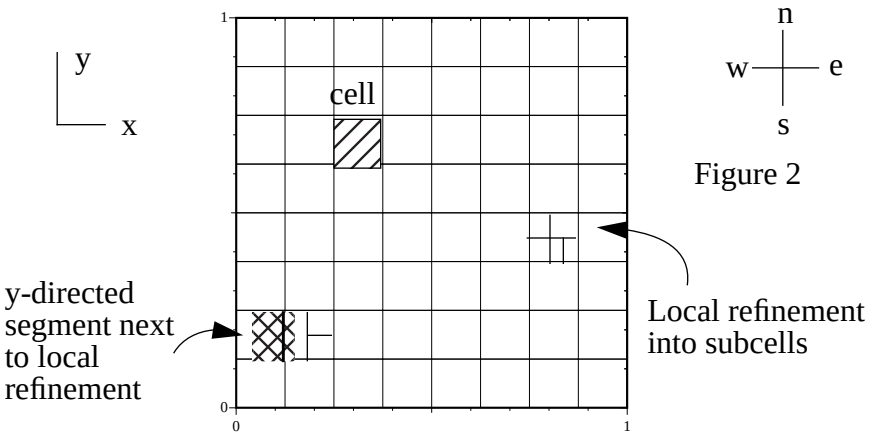


Figure 2

To accomplish local refinement, a cell can be divided into subcells. After refinement, all line segments will correspond to a segment of conductor whose current direction is along the length of

the segment as before. The width of all the segments must cover all of space occupied by the plane for the discretization to represent a solid plane. For this reason, the width of a segment extends halfway into each of its adjacent cells. As shown in the figure, a y-directed, or north-south, segment has a width which extends halfway into each of the cells to its east and west.

Internal Representation of the discretization

The representation shown in Figure 2is internally represented as a hierarchical tree of the cells. For simplicity in coding, the tree must be a binary tree. For example, given a simple discretization such as

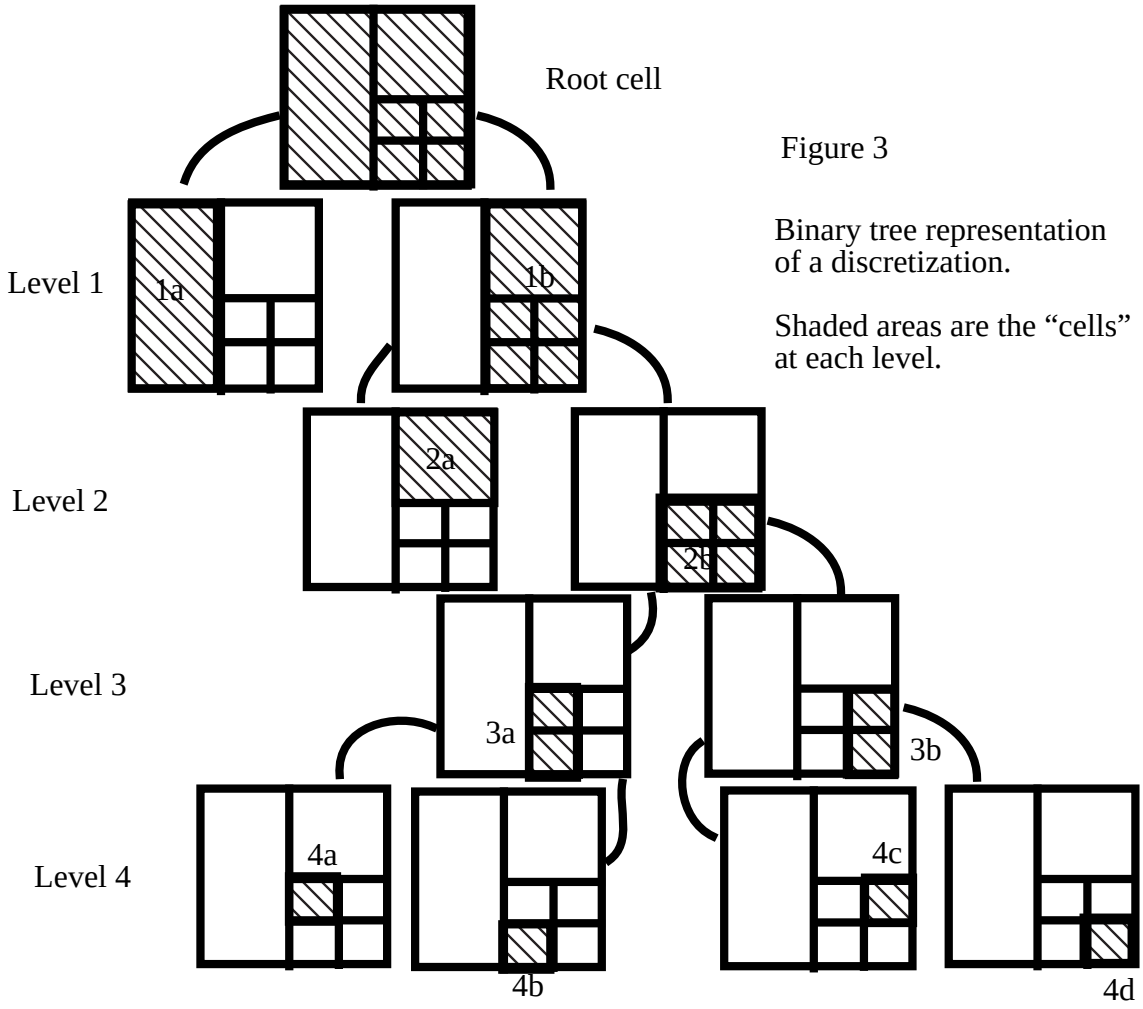


Figure 3

Binary tree representation of a discretization.

Shaded areas are the “cells” at each level.

that at the top of Figure 3, the entire plane is represented by one single cell as shown by the root cell in the figure. It’s two children are formed by dividing it into two east-west portions. The west portion, shown as the left child, is not discretized further and is a leaf cell in the tree at level 1. At level 2, there are two cells resulting from the north-south division of the east cell at level 1. The north cell is a leaf cell since it is not further discretized.

The restriction of a binary tree is not necessary and in fact the root cell can have a 2D grid of children

which will be necessary for defining meshed planes later in this document.

This description leads to a simple data structure with the following data structure::

```
/* a node in the tree */
typedef struct gcell {

  int index;            /* an index for debugging */
  void *children;      /* a pointer to a structure representing the subcells*/
  char children_type;    /* what is the structure for the children */

  struct gcell *parent; /* The parent of this cell */

  /* in the root cell coorinate frame, here are two corners of the cell */
  double x0, y0;    /* corner closest to origin (SW corner) */
  double x1, y1;    /* corner farthest from origin (NE corner) */

  union {
    struct g_edges *edges[NUMEDGES]; /* struct describing the edge of a cell */
    struct g_nodes *nodes[NUMNODES]; /* nodes of cell (only if leaf!) */
  } bndry;

} Gcell;
```

Here, the essential element is `children` which points to a structure representing the children of the current cell. As stated before, the only allowed structure for children is that of type Bi, for two children creating a binary tree:

```
/* a binary tree */
typedef struct bi {

  struct gcell *child1;  /* either North or East child */
  struct gcell *child2;  /*    South or West */

  char type; /* divided North/South or East/West */
#define NS 1
#define EW 2

} Bi;
```

Gcell's `children` element will be cast to type `Bi`. The structure contains pointers to the two children cells, plus `type` to describe whether the current cell is divided in the north-south direction or east-west.. The other elements of `Gcell` will be described later.

The hierachy can be input in a simple line by line format (described in Part 3), where each line specifies one cell, it's children, and whether it's divided east-west or north-south. The format of the file is:

```
<number of cells>
<index> <child-type> [<binary-type> <east/north child index> <west/south child index>]
<index> <child-type> [<binary-type> <east/north child index> <west/south child index>]
<index> <child-type> [<binary-type> <east/north child index> <west/south child index>]
    .
    .
    .
```

where the first line must be the total number of cells, and then the next <number of cells> lines describe the cell. index is the index of the cell, child-type is the type of child, either NONE for a leaf cell or B for binary. If childe-type is B, then binary-type must be EW or NS depending on whether the parent is divided into east-west sections or north-south sections. The children indices then follow in the order east-west or north-south. An example is provided in Part 3 of this document.

As will be described as Part 2 of this document, the user can also specify no discretization, but instead specify conditions on the discretization, such as: in a given region of the plane, there should be no cell greater than some given dimensions. In this case, the code will automatically refine the hierarchy. Or, both can be done in which an initial discretization is given by the user and then the code can further refine. Once the hierarchy is read in (in function readTree() in file read_tree.c, called from process_plane() which is called by the FastHenry function readGeom()), various information must be determined regarding adjacency of cells in order to construct the conductor segments for fasthenry. That process is described in the next section.

Determining Segments

Given a discretization, as described above, it is then necessary to generate the conductor segments for simulation in FastHenry. Each edge of the discretization of Figure 2 corresponds to one conductor segment and the width of that segment depends on the cells which share that edge as shown in Figure 2. The simple hierarchical description above does not contain any adjacency information however. For that reason, we define *nodes* of cells to be the corners of the rectangle defining the cell. Since every cell which is not a leaf cell has children who will share the parent's nodes, we avoid the replication by only allowing leaf cells to have direct pointers to nodes. This is Gcell's nodes element inside the bndry union. Only non-leaf cells can have edges (described later) which is why they can share a union structure to save memory. In addition multiple cells will share nodes as shown in Figure 4 where cells 1a, 2a, and 4a all share a node. The process

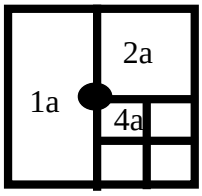


Figure 4

of determining this node sharing is performed by the function resolve_nodes() which is called by process_tree() in the file read_tree.c. First, every leaf cell is given a set of four nodes which is highly redundant, and then resolve_nodes(), from bottom up, determines when nodes are shared, and discards redundant nodes. The edges structure of Gcell is used as a representation of the edges of a non-leaf cell

to aid in determining shared nodes for cells at different hierarchical levels.

Side note: I realize in retrospect that there might be a better approach to determining this node information. resolve_nodes() is a bottom up approach in which a leaf cell propagates its information to its parent who puts together the information about its two kids. A less complex approach might have been to propogate the information from the root cell downward.

The node structure is `G_nodes` and contains all the adjacency information necessary to construct the conductor segments:

```
typedef struct g_nodes {

   int index;      /* an index for debugging */
   double x, y;
   struct gcell *cells[NUM_N_CELLS];
   /* need all four because of z-directed segs */
    struct g_nodes *adjacent[NUMADJ];
   char flag;          /* flag to mark things */
   double x_shift, y_shift; /* shift for center of z-directed segments */
   /* the number of segs is determined by gp->num_z_pts */
   SEGMENT **e_segs;  /* array of segs in east direction */
   SEGMENT **n_segs;  /* in north direction */

   struct g_nodes *prev;
   struct g_nodes *next;

} G_nodes;
```

`x` and `y` are the x and y coordinates in the relative coordinate system of the plane. `cells` are pointers to the cells to the north-east (cell 2a in Fig 4), south-east (4a), south-west (1a), and north-west(1a). `adjacent` are pointers to the adjacent nodes in the north, east, south, and west directions. Note that for Figure 4, the west node would be NULL. `x_shift` and `y_shift` will be determined later for the calling fasthenry with the fasthenry node points which must be in the center of the segment. `e_segs` and `n_segs` are arrays pointing to the fasthenry segments which run east and north from this node, respectively and will be filled by `generate_segs()` described below. `prev` and `next` are for maintaining the global linked list of nodes.

Given all the nodes and their adjacency information, the segments are formed by the function `generate_segs()`.

Incorporation into the FastHenry Algorithms

The forming of the segments is equivalent to deciding on the resistance and inductance values of the equivalent circuit. Once they are formed, a circuit solve is done to determine the port behavior of the conductor system. To solve the circuit, FastHenry uses the classic technique known as mesh

analysis. This involves enforcing Kirchhoff's Voltage Law around every "mesh" or loop in the circuit. FastHenry has the capability to search the graph generated by the network of conductors to find all necessary meshes, however would be very inefficient for mesh determination for such an interconnected network as a discretized plane. A priori, we can see that for any discretization as we've defined above, the boundary of every leaf cell corresponds to exactly one mesh in the circuit and thus it is straightforward to determine these meshes. This is done in `make_nonuni_Mlist()`.

Additionally, if the user defines circuit conditions by connecting user-defined segments to the plane or shorting points of the plane together with `.equiv` statements, then a mesh must be formed as a path between two user-defined connection points. The path of segments connecting two points of the plane is searched for with the function `path_through_nonuni_gp()` which calls the recursive function `get_a_nonuni_path()`.

Part 2: Creation of appropriate discretizations

The last section described the implementation of nonuniform discretizations. This section describes how to specify or create a nonuniform discretization.

FastHenry can read in any discretization specified as a binary tree, thus the process of grid generation could (and should?) be done by a completely separate program. However, in order to test the nonuniform discretization routines, a simple library of utilities was developed to generate a discretization internally. This library does not generate optimal discretizations and it is recommend that the user check the generated mesh for appropriateness.

This section first describes specifying a binary tree explicitly through an external file and then describes the library of routines for generating a discretization. Then the new formats for making external contacts to the plane and specifying meshed groundplanes are described. Note that Part 3 of this document contains many examples of the utilities described in this section and should be consulted while reading this section.

Specifying a plane in FastHenry

An example of a FastHenry version 2.0 specification of a simple plane (see the FastHenry 2.0 User's Guide) is:

```
g_the_plane x1=0 y1=-2 z1=0 x2=10 y2=-2 z2=0 x3=10 y3=2 z3=0 thick=0.01
+ seg1=20 seg2=20
+ ng1 (1,0,0)
+ ng2 (9,0,0)
```

where `g_the_plane` is the name of the plane, p1 = (x1,y1,z1), p2 = (x2, y2, z2), p3 = (x3,y3,z3) specify 3 corners of the rectangle defining the plane, `thick` is the thickness of the plane. ng1 and ng2 are connection points to the plane, and seg1 and seg2 specify the uniform discretization.

In the new FastHenry 3.0, to specify a nonuniform plane, do not specify seg1 and seg2, but instead specify either

- **A file containing the nonuniform description,**
- **Regions of the plane to refine at run time.**
- **or, both**

For instance, a file specification might look like:

```
g_the_plane x1=0 y1=-2 z1=0 x2=10 y2=-2 z2=0 x3=10 y3=2 z3=0 thick=0.01
+ file=trace.nonuni
+ contact equiv_rect ng1 (1,0,0,0.05,0.05)
+ contact equiv_rect ng2 (9,0,0,0.05,0.05)
```

In this case, FastHenry will look for the file “trace.nonuni” in the current directory for a description of the nonuniform discretization described in Part 1 of this document. Contacts to the plane should have dimensions and the contact region should be at an equipotential. This is specified with the new “equiv_rect” contact keyword which defines contacts at (1,0,0) and (9,0,0) to have dimensions 0.05x0.05.

If the specification `file=NONE` is given, then no file is to be specified and the hierarchy is a single root cell. This is useful if the discretization is to be done at run time as described in more detail in the the section “Specifying discretization to be done at run time”. For instance,

```
gpower x1=-5700 y1=-2666 z1=150 x2=2800 y2=-2666 z2=150
+      x3=2800 y3=10834 z3=150 thick=20 file=NONE
*
+ contact connection nx_power0_1 (-3700,0,129,50,50,2.5)
+ contact connection ny_power0_1 (0,8800,129,50,50,2.5)
*
* Define initial meshed grid
+ contact initial_mesh_grid (34, 54)
*
* refinement under the signal traces for high freq
* the x-directed traces
+ contact trace (-3700,0,150,-1700,0,150,50,1)
* the y-directed traces
+ contact trace (0,1700,150,0,8800,150,50,1)
* the diagonal trace
+ contact trace (-1700,0,150,0,1700,150,50,5)
```

This excerpt from an example in Part 3 defines a plane with two external connections, `nx_power_01` and `ny_power0_1` each of width 50x50 defined with the “contact connection” utility. There is a trace running near the surface of the plane and this is communicated to the discretization routines through the “contact trace” utility. After reading in such a specification, FastHenry will discretize the plane finely near the connection points to accurately model the resistance of the plane, and also discretize finely underneath the traces to capture the current crowding that will occur at high frequencies.

Also, p1, p2 and p3 still specify the dimensions of the plane, but also p1 specifies the origin of the plane coordinate system which will be referred to later in this document. The vector from p1 to p2 specifies the the x-direction in the plane coordinate system. And similarly, the vector from p2 to p3 specifies the y-direction.

Specifying contacts to the plane

In version 2.0, contacts to the plane were specified as a single point, such as

```
+ ng1 (0,4,2)
```

However, a contact is not a single point but has nonzero dimensions. To properly model a contact, we must account for its dimensions. Empirically, we have found that accurate modeling of the contact area is important for accurate resistance computation, but its impact on inductance is not as strong.

All contacts in FastHenry 3.0 must be modeled as rectangles. The area of the rectangle will be forced to be an equipotential region labeled with a single name with the "equiv_rect" utility:

```
+ contact equiv_rect the_name (x,y,z,x-width,y-width)
```

for instance

```
+ contact equiv_rect n_the_name (10,4,2,26,30)
```

This defines the contact named "n_the_name" (must start with an "n") at the location on the plane (10,4,2) with dimensions (26,30) where 26 is the width in the x-direction and 30 in the y direction. It could then be referred to later in the input file as regular node such as with

```
N_other_node x=3 y=4 z=5
.equiv N_other_node n_the_name
```

The "contact connection" utility is similar to the equiv_rect utility but also affects the discretization and will be discussed in the next section.

Specifying discretization to be done at run time

After an initial discretization specified with the `file=` keyword, the user can specify further refinement to be done at run time with the "contact" keyword as demonstrated in the above examples. FastHenry will make an educated guess as to an appropriate discretization based on paramenters given by the user. The name is slightly misleading in that the contact keyword is for making both a contact *discretization* in addition to an actual connection to the plane. For instance, the `contact decay_rect` keyword in the following plane specification forces the dimensions of all cells contained in a rectangle of dimensions (0.05,0.05) at the point (1,0,0) to be no larger than (0.015,0.015). The discretization also gradually coarsens outside the rectangle (hence the word decay). This example will be described in more detail later.

```
g1 x1=0 y1=-2 z1=0 x2=10 y2=-2 z2=0 x3=10 y3=2 z3=0 thick=0.01 file=NONE
+ contact equiv_rect ng1 (1,0,0,0.05,0.05)
+ contact equiv_rect ng2 (9,0,0,0.05,0.05)
+ contact decay_rect (1,0,0,0.05,0.05,0.015,0.015,2,2)
```

The `decay_rect` contact utility, in addition to the `connection` and `trace` utilities, are built upon many simpler utilities (all of which are contained in the source file `contact.c`). This section describes these utilities starting with the most basic and working upward. Please see Part 3 for figures which illustrate these examples.

The `point` utility discretizes around a point. The `point` utility forces the cell that contains a point to be divided until the point is contained in a cell no bigger than the specified dimensions. In the following example, after the point utility is finished, the point (1,0,0) will be contained in a cell of dimensions no bigger than (0.1,0.2) where 0.1 is the width of the cell in the plane coordinate system's x-direction, and 0.2 is the width in the y-direction:

```
+ contact point (1,0,0,0.1,0.2)
```

Similarly, the `line` utility forces all cells along the given line to have dimensions no greater than the specified dimensions. The line utility walks along the line, calling the point utility for every cell it crosses. For instance, the following specifies that no cells along the line from (1,0,0) to (9,0,0) are greater than (0.1,0.2):

```
+contact line (1,0,0,9,0,0,0.1,0.2)
```

The `rect` utility forces all cells contained in the given rectangle to be no bigger than (0.1,0.2). It calls the line utility for a set of parallel lines in the x-direction and another set in the y-direction to cover the rectangle. The following forces all cells to be no bigger than (0.1, 0.2) inside the rectangle centered at (5,0,0) with dimensions (2,2):

```
+ contact rect (5,0,0,2,2,0.1,0.2)
```

The `decay_rect` utility is similar to the rect utility except that it also forces that the cells outside of the rectangle to gradually coarsen as the distance from the rectangle grows. This is accomplished by calling the rect utility for gradually larger rectangles with gradually larger cell sizes. The specification described at the beginning of this section is an example, where, additionally, the final (2,2) makes the utility stop once the cell size constraint is greater than (2,2).

The decay_rect utility chooses the next rectangle and cell size assuming that there is a source of current inside the original rectangle and that we wish to insure that the FastHenry segments will carry the same net current as we move away from the rectangle. To explain the approach in more detail, we first assume that the current density decays as 1/r where r is the distance from the contact center. We handle x and y directions separately.

Let the description of the parameters for decay_rect be

```
+ contact decay_rect (x0, y0, z0, 2*xr, 2*yr, 2*xc, 2*yc, 2*xs, 2*ys)
```

Consider the x-direction. As defined above, half the rectangle width is xr and half the maximum cell width is xc. Let r0 = xc/xr. Then the current in the cells on the outer edge of the rectangle is roughly

$$I = \int_{(xr-xc)}^{xr} \frac{1}{r} dr = \int_{xr\,(1-r0)}^{xr} \frac{1}{r} dr = \ln\left(\frac{1}{1-r0}\right)$$

To match this current outside the xr rectangle, we choose a new width, xr_new = xr*1/(1 - r0) , and thus a new cell width xc_new = xr_new - xr. So outside the rectangle

$$I = \int_{xr}^{xrnew} \frac{1}{r} dr = \int_{xr}^{xr\left(\frac{1}{1-r0}\right)} \frac{1}{r} dr = \ln\left(\frac{1}{1-r0}\right)$$

The process continues using a fixed r0. The `rect` utility is called for each xr_new and xc_new until xc_new is greater than the maximum specified by xs. An identical approach is used for the y-direction. Note that the r0 must be less than 1. For r0 close to 1, the discretization will coarsen very quickly and for r0 closer to 0, the coarsening will be gradual.

Grouped contact utilities

In order to properly make a contact to a plane with appropriate discretization, first, the plane must be discretized near the contact, and second, the contact region must be forced to be an equipotential. These two contact utilities are combined into one “contact connection” utility.

```
+ contact connection the_name (x,y,z,xwidth,ywidth,ratio)
```

which is equivalent to

```
+ contact decay_rect (x,y,z,xwidth,ywidth,xwidth/ration,ywidth/ration,-1,-1)
+ contact equiv_rect the_name (x,y,z,xwidth,ywidth)
```

Also, it is important to discretize finely underneath a trace or any current carrying element close to the plane. But it is not necessary to do so in the direction of the trace, only in the orthogonal direction. This can be accomplished roughly with

```
+ contact trace (x0,y0,z0,x1,y1,z1,trace_width,scale_factor)
```

where the PROJECTION of the trace onto the plane goes from (x0,y0,z0) to (x1,y1,z1) and the trace has width trace_width. scale_factor is a factor to magnify the size of the cells under the trace when the trace is not parallel to x or y. The scale_factor reduces the number of fastHenry elements needed to model the trace at the cost of accuracy. Values for scale_factor are investigated in the `examples/trace.tests.release` file. Please see this file.

The trace utility just calls “contact line” a few times (maybe not enough!) to insure the cells underneath are no bigger than trace_width/2 in the direction of the trace width. The cells adjacent to the projection of the trace onto the plane are refined to be no bigger than trace_width. (Maybe more is needed?)

The trace utility can be replaced with decay_rect when the traces are along x or y.

Example:

```
+ contact trace (-1,-1,0,1,1,0,0.01,10)
```

discretizes for a trace whose projection is from (-1,-1,0) to (1,1,0) with width 0.01. The cells directly under the trace are no bigger than 10*0.01/2

Note scale_factor has no effect when the traces are parallel to the x or y axis and the scaling varies continuously from 1 to scale_factor as the angle of the trace goes from 0 to 45 or 90 to 45.

Another example:

```
+ contact trace (-1000,0,0,1000,0,0,10,1)
```

will do

```
+ contact line (-1000,0,0,1000,0,0,2000,5)
+ contact line (-1000,5,0,1000,5,0,2000,5)
+ contact line (-1000,-5,0,1000,-5,0,2000,5)
+ contact line (-1000,15,0,1000,15,0,2000,10)
+ contact line (-1000,-15,0,1000,-15,0,2000,10)
```

Note that there is no discretization in the x dir as desired since 2000 is the length of the trace.

In the first example of a 45 degree trace, there is no disrectization “direction” along the trace, so we are forced to discretize in both x and y which can lead to many cells. The scale_factor is used to reduce that by magnifying the last two values sent to “contact line” by (scale_factor^fabs(tan(theta)) where theta is the angle of the trace relative to the x axis and fabs is the absolute value.

Meshed planes and Mimicking a uniform discretization

There is no facility in the nonuniform implementation for specifying individual physical holes in an otherwise solid plane (the facility exists for uniformly discretized planes). However, the user can specify a regular grid of rectangular holes to model a *meshed* plane.

The plane can be initially divided into a uniform grid of cells which are not a power of two as would be required by the binary tree representation. With this facility, we can mimic the uniform discretization routines. The plane can be initially broken into a grid of cells by specifying

```
+ contact initial_grid (10,12)
```

The plane would be broken into a 10x12 grid of cells BEFORE any other refinement is done with other contact utilities. 10 is the number of rows and 12 the number of columns where a row has a contant y value. Thus, the old style uniform plane specification:

```
seg1=10 seg2=12
```

could be replaced with

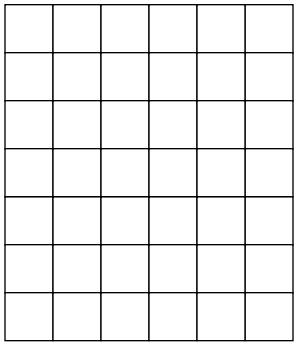
```
file=NONE contact initial_grid (10,12)
```

One advantage of using the new nonuniform specification is that there is no overhang of the edge segments.

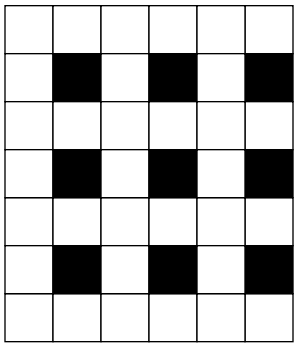
If the initial grid is to be meshed, that is, have an array of holes in it,

```
+ contact initial_mesh_grid (10,12)
```

will break the plane into a 10x12 grid, and then mark as a “hole” every cell that has an even value for both of its indices where the numbering is from the top left. For instance, the grid generated by “`contact initial_grid (7,6)`” would look like:



and a grid generated with "`contact initial_mesh_grid (7,6)`" will be



where a black square represents a hole. No conductor will be defined in that square region. Further local discretization can be accomplished with other `contact` utilities as described earlier in this section however there is no utility for discretizing more finely around every hole.

What's missing: Breaking segments to maintain multipole accuracy

The implementation of the multipole approximations assume that it's a reasonable approximation to consider all the current associated with a segment to come from the center of the segment (a "point" charge). For very long, thin segments this is not true. The original implementation of FastHenry will break segments in half along their length if judged too long. Something equivalent should be done for nonuniform planes but is not in this release. This will result in warnings such as

```
DivideSegs: Warning: tried to divide an indivisable segment.
  Segment length: 0.000750,  maximum allowed length: 0.000379  The segment is probably
part of a ground plane.
  If so, decrease the partitioning level by 1 or refine the ground plane
```

For problems where all refinement is done gradually, that is no sharp changes from fine to coarse discretization, then the results do not seem to be greatly impacted.

Part 3: Examples

This section gives examples of running FastHenry and generating and viewing discretizations. All examples should be in the `examples` directory.

Predefined discretization

The following is an example of using a predefined discretization specified in the file `examples/tree_sample.hier`.

```
43
1     B     EW    2     3
2     B     EW    4     5
3     B     EW    6     7
4     B     NS    8     9
5     B     NS    10    11
6     B     NS    12    13
7     B     NS    14    15
8     NONE
9     B     NS    16    17
10    NONE
11    B     NS    18    19
12    NONE
13    B     NS    20    21
14    NONE
15    B     NS    22    23
16    NONE
17    NONE
18    B     NS    24    25
19    NONE
20    B     EW    26    27
21    NONE
22    NONE
23    NONE
24    B     EW    28    29
25    B     EW    30    31
26    B     NS    32    33
27    B     NS    34    35
28    NONE
29    NONE
30    NONE
31    B     EW    36    37
32    NONE
33    B     NS    38    39
34    NONE
35    NONE
36    B     NS    40    41
```

```
37    B     NS    42    43
38    NONE
39    NONE
40    NONE
41    NONE
42    NONE
43    NONE
```

The input file that uses this plane is `examples/tree_sample.inp` shown below and defines a plane in the y-z plane with its center at (10,10.5,10.5). This was used for testing and doesn't represent a good discretization. See the FastHenry manual for details.

```
*  A fasthenry input file for a nonuniform ground plane

.units m
.default sigma=5.8e7

g1 y1=10 z1=10 x1=10 y2=11 z2=10 x2=10 y3=11 z3=11 x3=10
+ relx = 10
+ file=tree_sample.hier
+ thick=0.1
+ nin (,10.1,10.1)
+ nout (,10.9,10.5)
+ n3 (,10.56,10.31)
+ n4 (,10.5,11)

.external nin nout
.external nin n4
.external n3 nout

.freq fmin=1 fmax=1e9 ndec=0.1

.end
```

With fasthenry-3.0/bin in your path, this problem can be run with:

```
fasthenry tree_sample.inp
```

However since there are fewer than 1000 filaments in this problem, it is recommended that it LU decomposition be used instead of the multipole algorithm:

```
fasthenry -sludecomp -aoff tree_sample.inp
```

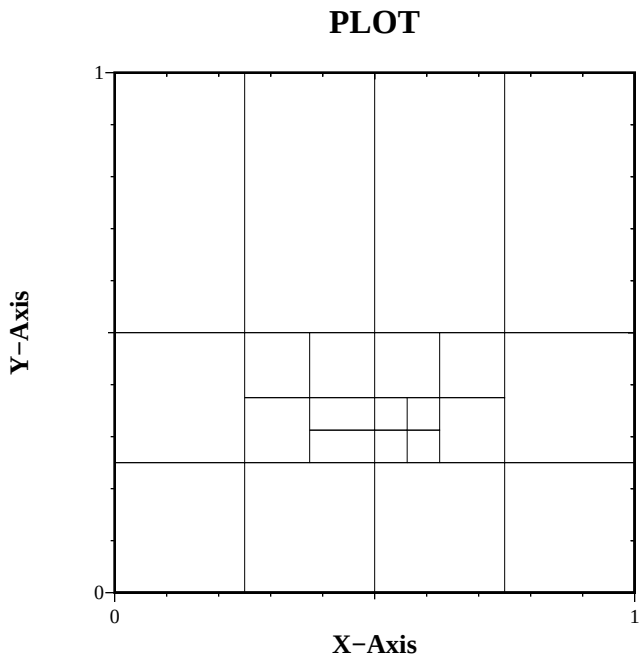
To view the plane discretization without solving for inductance:

```
fasthenry -f hierarchy tree_sample.inp
```

This produces the file hier.qui. This file (as well as the file `zbuffile` produced with the "-f simple") can

be viewed using the zbuf program as described in the FastHenry 3.0 manual.

The postscript output from zbuf should look something like:



Run time mesh refinement

To show examples of using the run time refinement capabilities, the following template will be used in this section with the `contact` line replaced as described in each section:

```
* Template for showing autorefinement
.units m
.default sigma=5.8e7

g1 x1=-10 y1=-10 z1=0 x2=11 y2=-10 z2=0 x3=11 y3=11 z3=0
```

```
+ file=NONE thick=0.01
+ contact xxxxxx (n,n,n,n,...)
+ n_in (0.0,0,0)
+ n_out (1.0,1,0)

.external n_in n_out
.freq fmin=0 fmax=1e9 ndec=1
.end
```

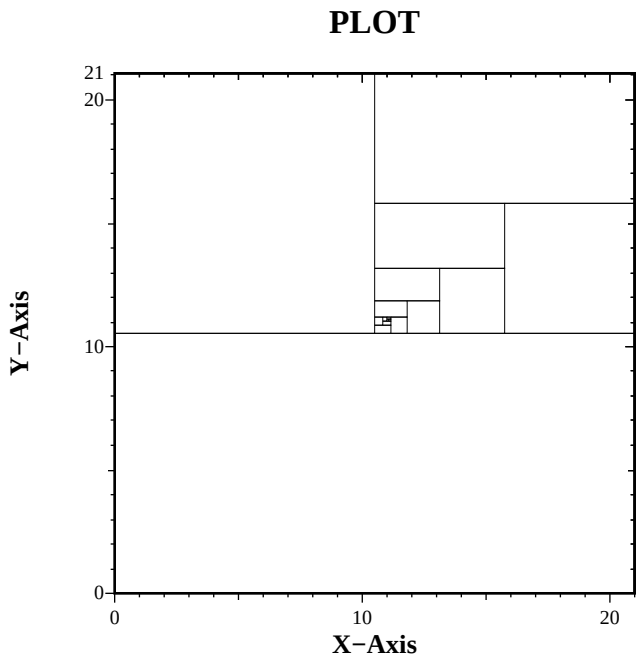
The files for these examples are in `examples/template.inp`.

Point contact utility

Replacing the contact line with:

```
+ contact point (1,1,0,0.1,0.1)
```

produces the following:

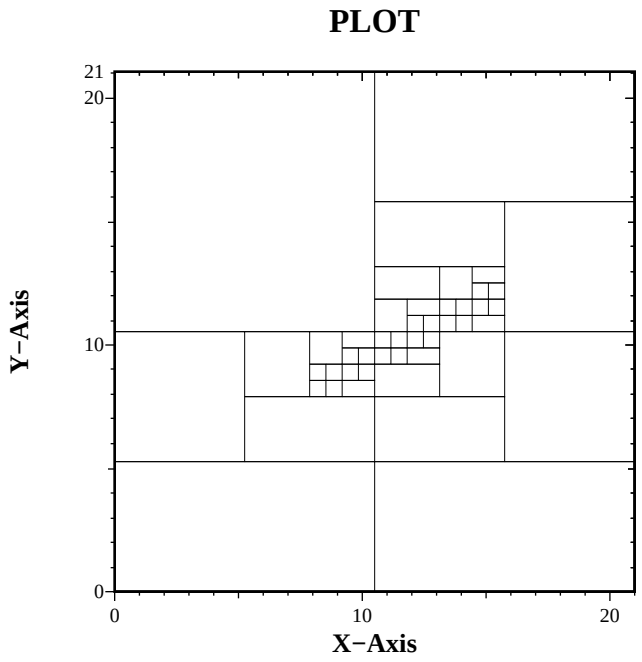


Line contact utility

Replacing the contact line with:

```
+ contact line (-2,-2,0,5,2,0,1,1)
```

produces the following:

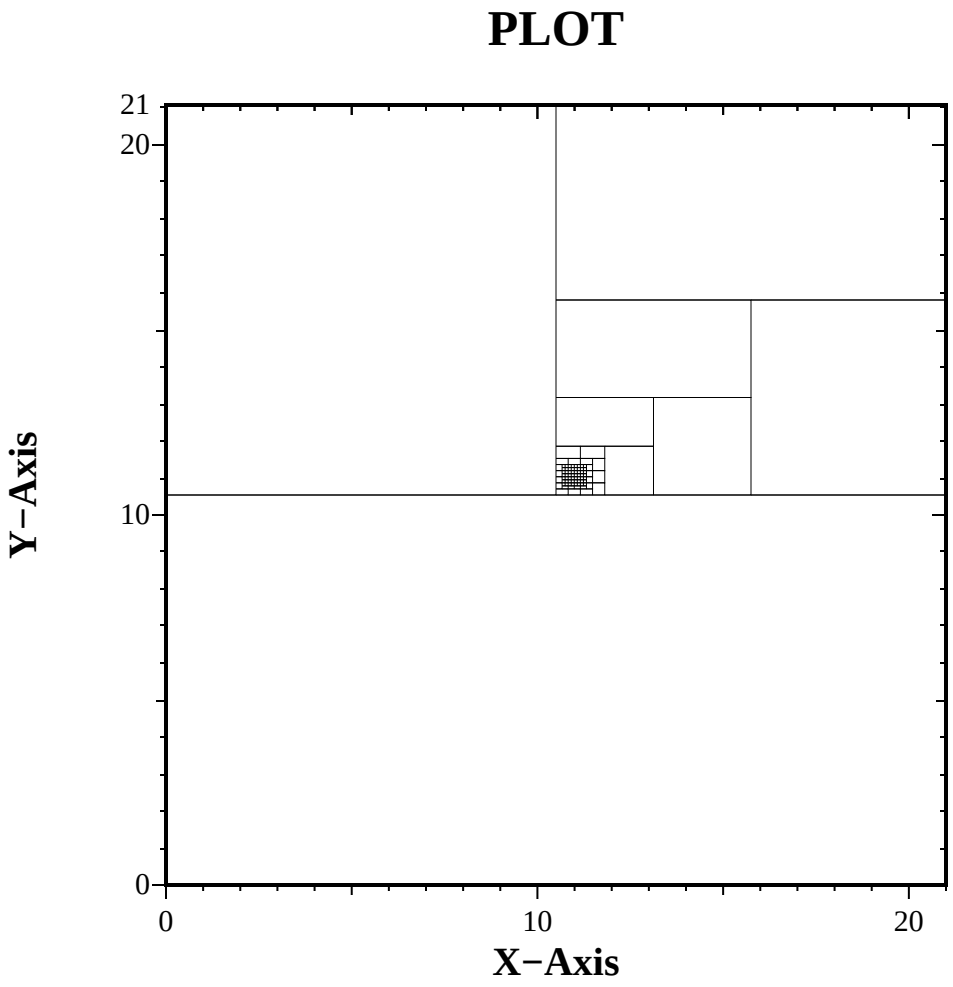


Rect contact utility

Replacing the contact line with:

```
+ contact rect (1,1,0,0.5,0.5,0.1,0.1)
```

produces the following:

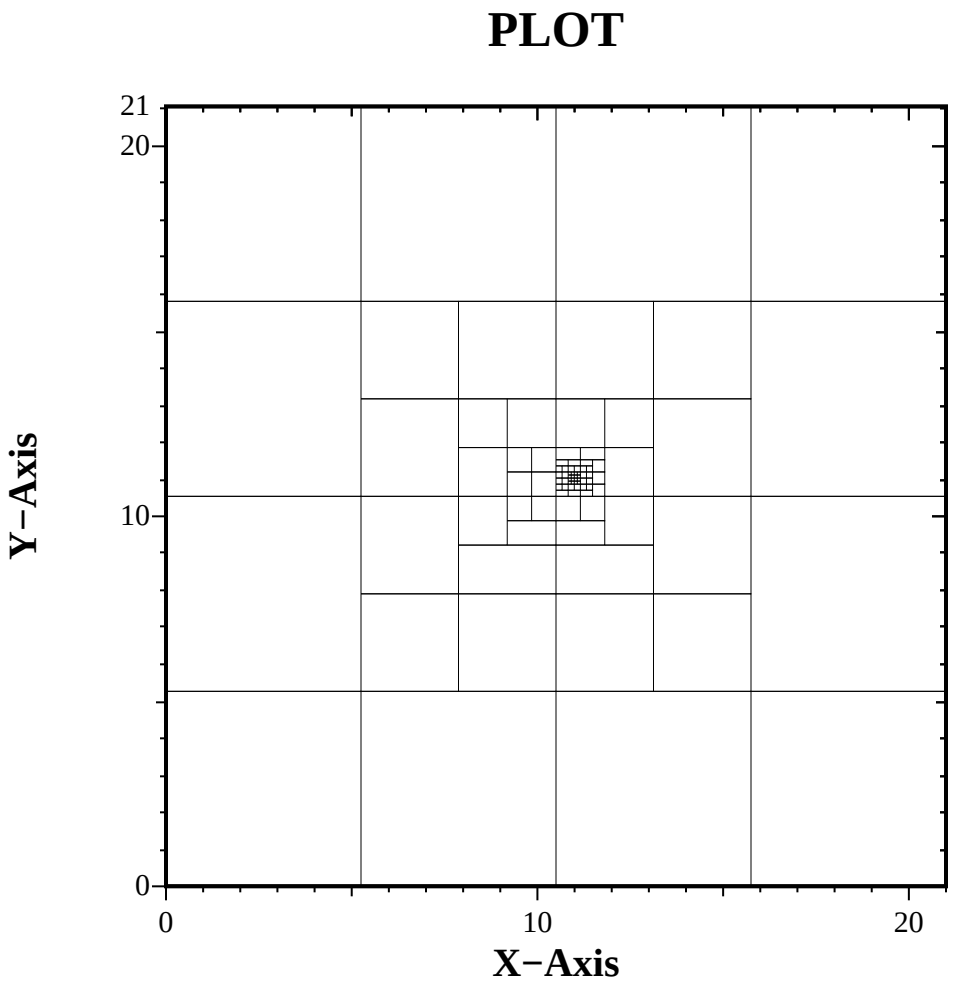


Decay_rect utility

Replacing the contact line with:

```
+ contact decay_rect (1,1,0,0.2,0.2,0.1,0.1,20,20)
```

produces the following:



Note that the xc/xr = 0.1/0.2 = 0.5. Choosing this ratio closer to 0 produces more gradual refinement. The opposite is true for the ratio closer to 1.

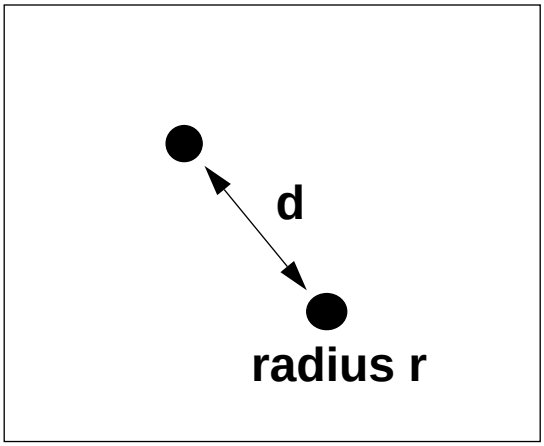
The highest level utilities: `connection` and `trace` are described at the end of the next section.

Benchmarks and accuracy testing

This section describes: testing the code for a resistance problem for which the analytic solution is known, a 2D problem of a single trace over a plane which can be be compared to the results of a 2D tool, and a 3D problem of a trace over a plane which can be compared to a 3D tool.

Resistance between contacts

For a thin plane, *infinite* in extent, the current distribution at DC does not vary in the z-direction. The resistance between two well separated cylindrical contacts



can be shown to be

$$R = \frac{1}{\pi\sigma t}\ln\left(\frac{d-r}{r}\right)$$

where t is the thickness of the plane and σ is the conductivity. Using r = 0.1, d = sqrt(2) and σ = 5.8e7 we get R = 2.71e-6. Using a very fine discretization for a 21x21 plane entered as

```
g1 x1=-10 y1=-10 z1=0 x2=11 y2=-10 z2=0 x3=11 y3=11 z3=0
+ file=NONE thick=0.01
+ contact decay_rect (0,0,0,0.025, 0.025,0.001,0.001,3,3)
+ contact decay_rect (1,1,0,0.025, 0.025,0.001,0.001,3,3)
```

defines a problem of 70,000 elements. Running FastHenry for a DC resistance problem gives R = 2.73e-6 which is less than 1% error. However since there is no utility yet for forcing the contact circles to be at an equipotential, 200 points around each circle had to be individually tied together. The code for generating the input file is examples/make_nonuniform.c. The executable takes one input parameter: the radius of the contact. A sample input file is given in the file examples/ nonuni01.inp. Running FastHenry on this file will produce lots of “trying to equiv points that are already equiv’ed” warnings but that is expected because the 200 points around each circle is an overkill to insure that all node points that exist around the circle are set to be the same potential.

The number of elements for only 1% accuracy is rather discouraging, however it has been observed that the inductance is not so sensitive to this discretization as will be shown in later sections. Also, the inability to come closer to the analytic solution may be from poor choice of the parameters for `decay_rect`. Or the plane may need to be extended to more accurately model an infinte plane. Also various attempts at choosing good xc and xr were tried and the results are given here. For fewer than 2000 elements, FastHenry gives R = 2.80e-6, and for one case, R = 2.72e-6 for 8000 elements which may be chance. This needs to be further investigated.

In the following very raw table, the “decay (<val1>,<val2>)” indicates the xr and xc values used in the decay_rect call.

```
radius = 0.05  xlength=ylength=21  ratio=1/3

1.85808e-06

radius = 0.01 xlength=ylength=21  ratio=1/3    decay (0.01,0.01/3)
  (1380? elements)

   2.85174e-06   (analytic: 2.7136e-06)

radius = 0.01 xlength=ylength=21  ratio=1/3    decay (0.03,0.01)
  (1350 elements)

   2.8576e-06

radius = 0.01 xlength=ylength=21  ratio=1/2    decay (0.02,0.01)
  (1000 elements)

  2.87275e-06

radius = 0.01...  decay (0.03,0.01) plus rect (0.025,0.003)
   1676 elements  (current looks like a circle! )

  2.80572e-06

radius = 0.01  same as above except xlength=ylength=41  (no change)
  1812 elements
  2.80961e-06

radius = 0.01 length=21    decay_rect=(0.025,0.003)
  8152 elements!  (file moved to nonuni_accurate.inp)
  2.72702e-06

radius.....   length=11  others as above
  7316 elements
  2.76528e-06

radius...  length=21 decay_rect=(0.025,0.001)  (need more equiv!)
  70436 elemnts
  2.74784e-06

radius...  length=21 decay_rect=(0.025,0.001)  200 equiv’s per contact
```

```
                              (all of above have 40)
  70436 elements
  2.733e-6       (1% error)   (seemed to take 1305 seconds while previous
                         took 348? or maybe i read it wrong before)
```

2D inductance comparison

The file `plane.in` is an example of using FastHenry to model a 2D inductance calculation of a line over a plane. The results were generated with

```
fasthenry -S _plane_2d -sludecomp plane.in
```

The computed values were L=9.81 nH/cm and R=37.5 ohm/cm which closely matched the values from two 2D simulators.

3D inductance comparison

For a single trace over a substrate the inductance and resistance at 330 MHz is computed with the input file `3d_example2.inp`

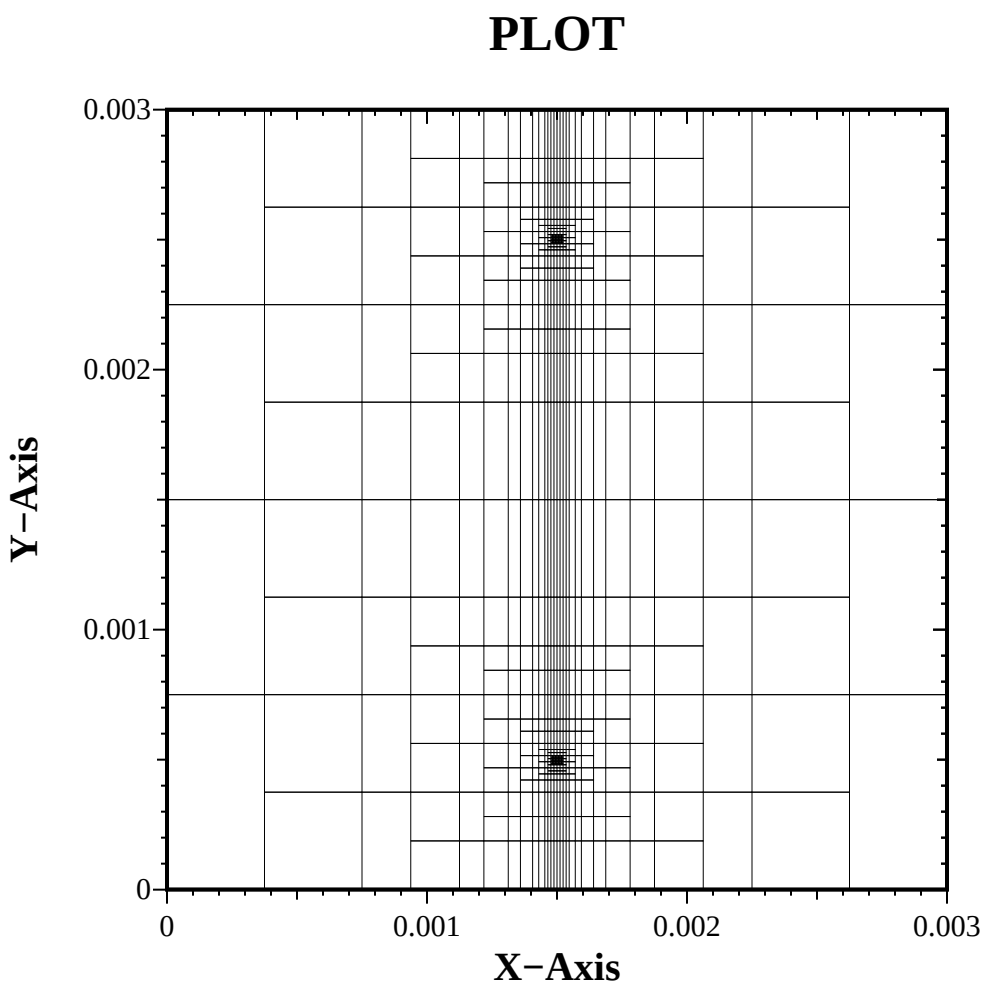
```
* a M5 line (26um wide, 1um thick) over Si substrate 8um below, sigma=2.6e7
* 430um thick substrate, sigma=1.5e4 1/(ohm*m)
* line length=1000um
.units uM
*
* Define substrate, sigma=1.5e4 1/(m*ohm)= 0.015 1/(um*ohm)
g1 x1 = -1500 y1 = -1500 z1 = 0
+  x2 = 1500  y2 = -1500 z2 = 0
+  x3 = 1500  y3 = 1500  z3 = 0
+  thick = 430  sigma= 0.015
+ file=NONE
*  under the trace
+ contact decay_rect (0,0,0,50,2000,13,3000,500,3000)
*  the contacts
+ contact decay_rect (0,-1000,0,30,30,10,10,3000,3000)
+ contact decay_rect (0, 1000,0,30,30,10,10,3000,3000)
+
+  nhinc = 3 rh=2
*     nodes to reference later:
+  np1 (0,-1000,0)
+  np2 (0,1000,0)
*
* line width=26um
* thick h=1um sigma=2.6e7 1/(m*ohm)=26 1/(um*ohm)
*
NS1 x=0 y=-1000 z=223.5
NS2 x=0 y=1000 z=223.5
```

```
EM5 NS1 NS2 w=26 h=1 sigma=26 nwinc=4 nhinc=1
.equiv NS2 np2
.external NS1 np1
*
.freq fmin=3.3e8 fmax=3.3e8 ndec=1
.end
```

Note that no forcing of the equipotential around the contact was done as it should. This leaves the contact as a single point with effective width equal to the segments that meet at the contact point, which in this case xc=10 and yc=10. Note also that under the trace, xc = 3000, essentially not restricting the width of segments in the x-direction. This is appropriate since current under a trace has large gradients in the y-direction but not along the trace in the x-direction. This generated a 4884 element problem which was solved in 348 seconds on an RS-6000-3BT consuming 80 MB of memory. The results were generated with:

```
fasthenry -S _3d_examp2 3d_example2.inp
```

And thus the results would be placed in Zc_3d_examp2.mat. The results from FastHenry give R = 3.75 ohm, L = 1.529 nH and L/2 = 0.764 nH. The plane discretization for this problem is shown below.



Viewing the current distribution

By specifying the "`-d grids`" option to fasthenry, various files will be produced for the viewing of the current distribution in space and in planes. With each segment is associated a 3D direction and complex current. The real part of current density for each of these segments is dumped to the file `Jreal.mat` which specifies an arrow pointing along the segment with magnitude proportional to the real part of the current density. Each line in the file is of the form

```
x y z vx vy vz
```

where (x,y,z) is the location of one end of the segment, and (vx,vy,vz) is the vector direction along the segment. In this case, the magnitude of the vector is proportional to the real part of the current density. Similarly, the imaginary part is dumped to `Jimag.mat,` and the magnitude to `Jmag.mat.`

Note that this is for ALL segments, not just ground plane segments. The “-d grids” also will plot the current (not current density) at every node in the hierarchy by using the current in the segment pointing north for the y-direction and east for the x-direction. It outputs it in the file `Gridn_n.mat` as described in the FastHenry manual for uniform planes, and for nonuniform planes, this file should be converted to your favorite vector drawing format by the user. Its format for nonuniform planes is:

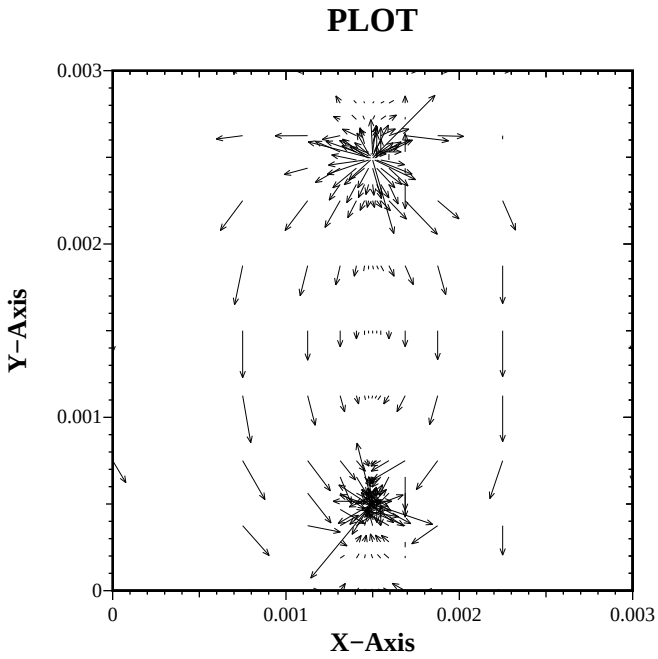
```
x y z xr +j xi yr + j yi
```

where (x,y,z) is the location of the vector and `xr+j xi` is the x-directed complex current and `yr + j yi` is the y-directed complex current.

Since the 3d example above has a much finer mesh than we need to observe, fasthenry was run on a coarser discretized problem in `3d_example2_coarse.inp`. Since it is difficult to make sense out of complex current, the frequency was set to 0 and the real part of the current was observed. FastHenry was run with

```
fasthenry -d grids -sludecomp 3d_example2_coarse.inp
```

LU decomposition was specified since the frequency is 0. The file Grid1_0.mat was then converted to an internal format used for vector drawing (basically awk was used to remove the +j xi and +j yi portions of the file). This yields the following plot:



Notice that since *current* and not current density is plotted the arrows for the current in the center of the plane are smaller than others along the x-direction. This is expected since the current in the x-direction is uniform, but the segments in the middle are smaller.

Using the `connection` and `trace` utilities

This is an example of using the highest level utilities: trace and connection. The `connection` utility creates an equipotential contact and also discretizes finely near it. The `trace` utility is used for telling the discretization routines that a trace is near to the plane and to discretize appropriately. The trace utility is most useful for traces that are not parallel to x and y since the `decay_rect` utility cannot be used. Here is an example of a trace sandwiched between a power and ground plane. The trace travels a short distance in x, some along a 45 degree bend and then a long distance in y. The

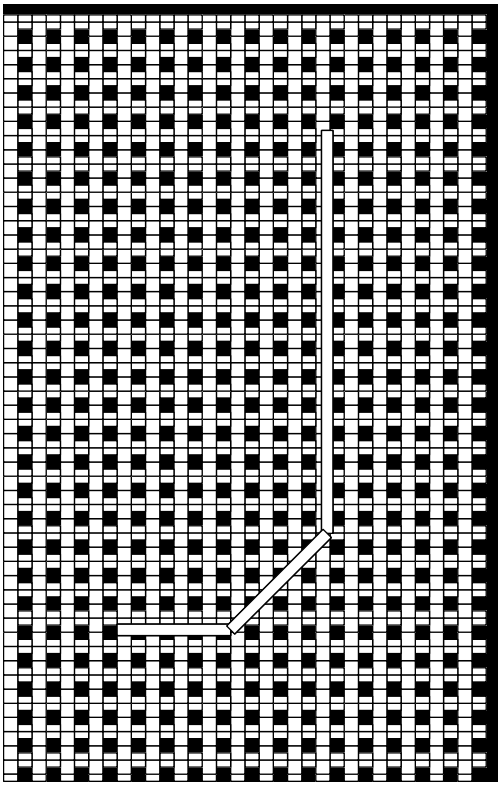
partial inductance of the trace, the power plane path over the trace, and ground plane path below the trace will be computed to give a 3x3 impedance matrix. The planes are meshed. This is example file `trace_over_mesh_new.inp`:

```
* A set of bending traces sandwiched between meshed planes
*  Units of microns
.units um

* default dimensions of traces
.default w=150 h=10

* do nodes on portion running along x-direction* starting x value is fixed for all the
traces along x
.default x = -3700 z = 86

nx0start y=0

.default x = -1700 z = 86
nx0end y=0

* we've done nodes, now make metal between them
ex0 nx0start nx0end

* Do something similar for traces running in y direction
.default y = 1700 z = 86
ny0start x=0

.default y = 8800 z = 86
ny0end x=0

ex0 ny0start ny0end

* Now do diagonal portion connecting x traces to y traces
exy0 nx0end ny0start

*The meshed plane on top
gpower x1=-5700 y1=-2666 z1=110 x2=2800 y2=-2666 z2=110
+      x3=2800 y3=10834 z3=110 thick=10 file=NONE
*
* do contacts to plane
+ contact connection nx_power0_1 (-3700,0,110,30,30,2.5)
+ contact connection ny_power0_1 (0,8800,110,30,30,2.5)
*
* Define initial meshed grid
+ contact initial_mesh_grid (34, 54)
*
* refinement under the signal traces for high freq
* the x-directed traces
+ contact trace (-3700,0,110,-1700,0,110,30,1)
* the y-directed traces
+ contact trace (0,1700,110,0,8800,110,30,1)
* the diagonal trace
+ contact trace (-1700,0,110,0,1700,110,30,5)
```

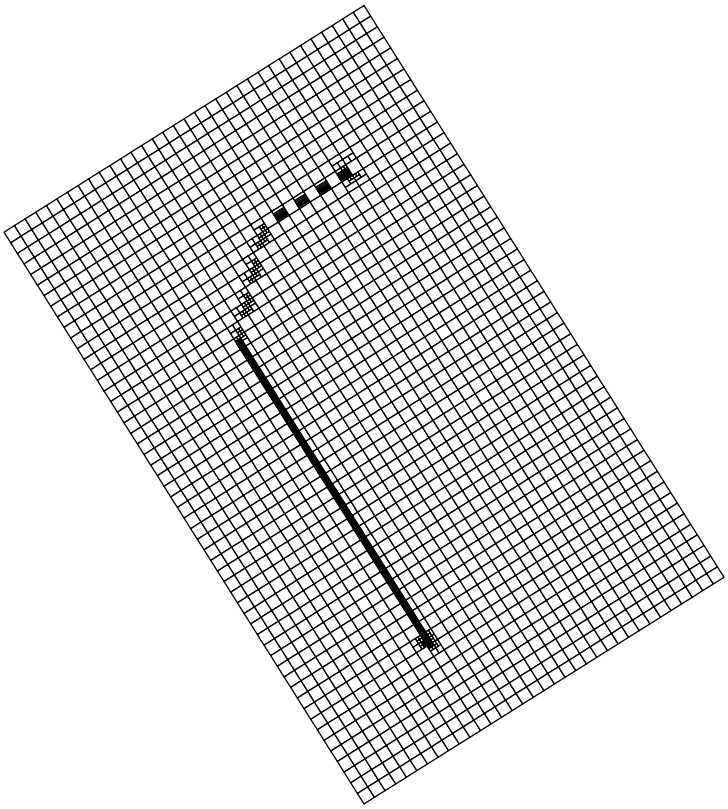
```
*The meshed plane on bottom
gground x1=-5700 y1=-2666 z1=0 x2=2800 y2=-2666 z2=0
+      x3=2800 y3=10834 z3=0 thick=13 file=NONE
*
* do contacts to plane
+ contact connection nx_ground0_1 (-3700,0,0,30,30,2.5)
+ contact connection ny_ground0_1 (0,8800,0,30,30,2.5)
*
* Define initial meshed grid
+ contact initial_mesh_grid (34, 54)
*
* refinement under the signal traces for high freq
* the x-directed traces
+ contact trace (-3700,0,0,-1700,0,0,30,1)
* the y-directed traces
+ contact trace (0,1700,0,0,8800,0,30,1)
* the diagonal trace
+ contact trace (-1700,0,0,0,1700,0,30,5)


* Define the ports
* The signal traces
.external nx0start ny0end signal_trace_0

* The power and ground paths
.external nx_power0_1 ny_power0_1 power_path_for_trace_0_and_1
.external nx_ground0_1 ny_ground0_1 ground_path_for_trace_0_and_1

* start at 125MHz and go up by factor of 2  (10^(0.3) ~= 2)
.freq fmin=125e6 fmax=500e6 ndec=3
.end
```

The geometry can be viewed by running FastHenry with the “-f simple -g thin” options and then running “zbuf zbuffile” to create the postscript zbuffile.ps. Actually, the follwing picture was created with “zbuf -m zbuffile” to produce a matlab file which can be viewed and printed from

matlab (see the FastHenry manual).



To view just the plane discretization: “`fasthenry -f hierachy trace_over_mesh_new.inp`” produces the file `hier.qui` which can be turned into the postscript file hier.ps with “`zbuf hier.qui -e0 -a0`” and looks like what is shown on the next page.



Note that this is only a picture of the discretization of one of the two planes. The initial_mesh_grid covers the plane with a large grid of cells first. Every other cell in this case is actually a hole. The presence of the trace causes most of the additional refinement on the plane. Note that the refinement under the 45 degree section and the short x section seems to have gaps. This is actaully because the cells where refinement seems to have been “skipped” are actually holes in the plane. Also note the extra refinement under the ends of the traces where connections to the plane are made.