SPACE 3D EXTRACTION MESH SUBNODE ASSIGNMENT PROBLEM

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 04-07
April 5, 2004

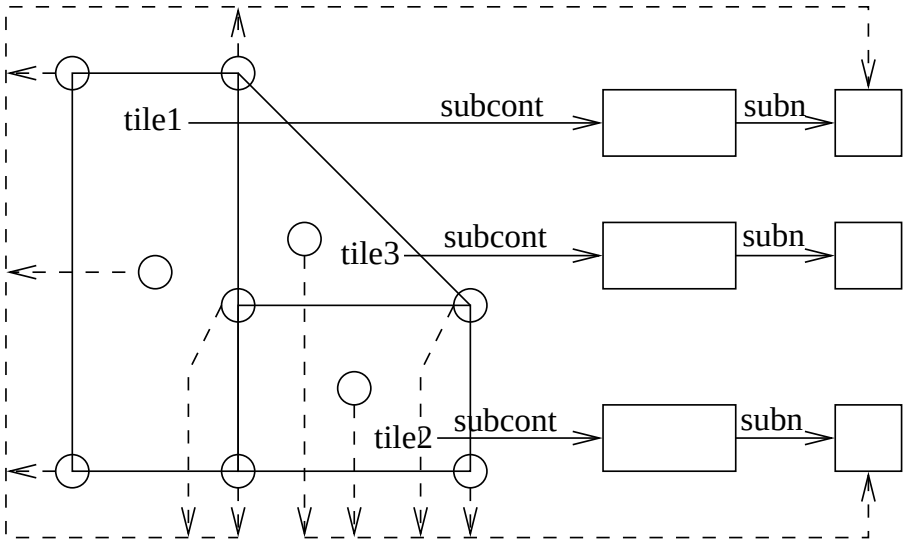
Last revision: August 24, 2004.

1. INTRODUCTION

The *space3d* program uses a 2D/3D mesh for accurate substrate resistance extraction (option **-B**) and for accurate capacitance extraction (option **-3C**). The mesh edges and points are modelled with spiderEdges and edge spiders. The spiders are unique, on a certain mesh (x,y,z) point can be only one spider. The edge spiders point to subnodes, and because edge spiders are unique, they can point only to one subnode. See the "Space 3D Substrate Extraction Application Note" (report EWI-ENS 03-04) and also the "Space Application Note About Nodepoints" (EWI-ENS 03-08) and also report "Space 3D Substrate Extraction by Using Makesubres" (EWI-ENS 04-06).

Thus, when on both sides of the edge is the same conductor, the spider can only point to one of these conductors. The *space3d* program shall use the subnode of the oldest tile (the left tile or the bottom tile).

When a center spider must be chosen, see function meshAddExtraSpider, this center spider must point to a subnode of that face. It happens, in special occasions, that no edge spider points to a subnode of that face. This happens by triangular tiles, when the upper right spider does not point to the tile→subcont→subn. See the following figure.



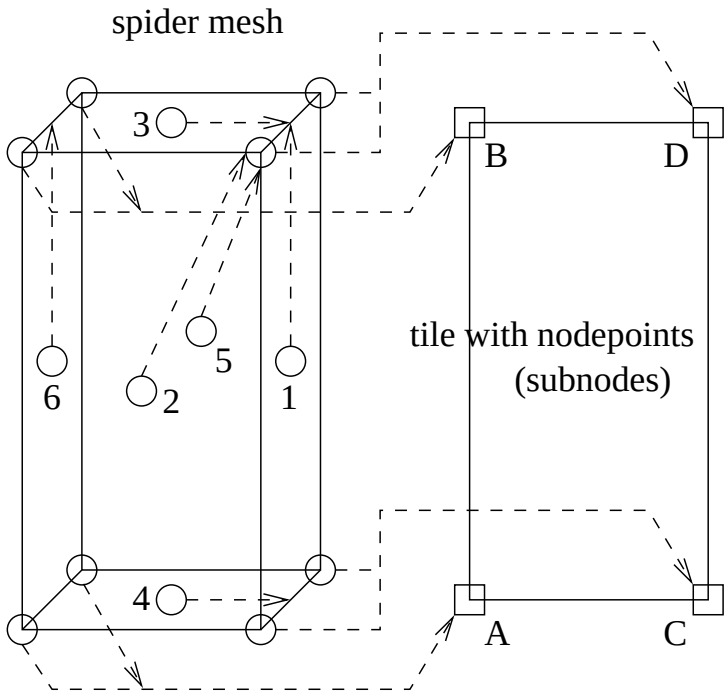
When tile3 is a separate substrate contact, for example in the distributed case, than never a value is assigned to tile3→subcont→subn.

To fix this problem for the substrate, i added member "sc_subn" to the face_t data struct. Now, function spiderFindFace sets "sc_subn" to the substrate tile subnode.

See appendix A for all changes made to the source files.

2. CAP 3D PROBLEM

In the extract pass is the 3D capacitance extraction done. Also in that case, the substrate nodes must be correctly connected to the capacitances. But what happens with the subnodes in the interconnect conductor? For example, we start with a simple rectangle, see figure below. We look only to the piecewise constant case, were center spiders are used.



We use 4 nodepoints (A to D), because optIntRes is "true" and it is a high res* conductor. The center spiders (1 to 6), who represent the faces of the conductor, point to the subnodes, which are in the nodepoints. Center spiders (1,2,3,5) point to the subnode of D. Center spider (4) points to the subnode of C. And center spider (6) points to the subnode of B. Thus, no center spider points to the subnode of A.

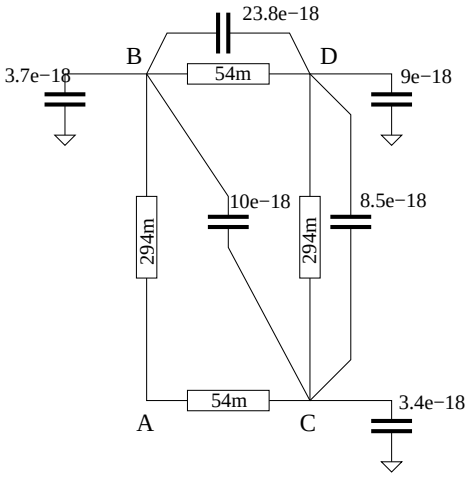
We use the following *space3d* command to extract 3D capacitors and interconnect resistors:

```
% space3d -r3C -E elem_t0.t -P params_t0 t0
```

Note that the fine network option **-%f** was not needed, because we used point terminals at every nodepoint position. See appendix B for the extraction results.

* Note that there are no couple caps by a low res conductor tile.

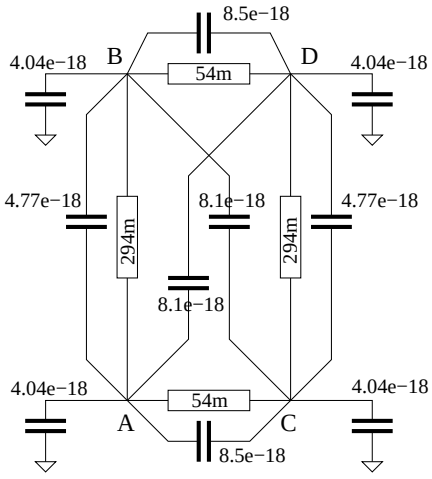
The 3D cap result is asymmetric, because node A gets no capacitance values assigned. The following figure gives a schematic picture of the 3D network result.



As can be seen, too many GND capacitance values are added to node D. However, there is also a problem with a number of couple capacitance values. These couple capacitance values are lost, because they cannot be assigned between the same subnode.
The lost values are calculated between the following spiders:

```
1-2 and 1-5 (see 6-2 and 6-5)
1-3    (see 1-4, 6-3 and 6-4)
2-3 and 3-5 (see 2-4 and 4-5),      2-5 (is maybe ok)
```

We can solve the above problems, by using another subnode assignment strategy. We can assign the calculated green value to the face corner points. To do this, we added the "cap3d.cap_assign_type" parameter. See also appendix C.
The following figure gives the schematic network for cap_assign_type 1.

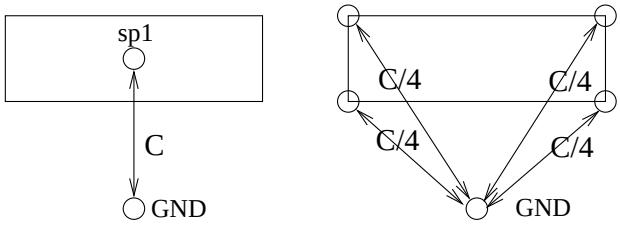


Note that cap_assign_type 2 does not omit spiders 2-5.

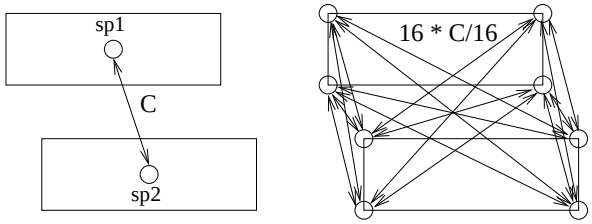
2.1 Result Statistics

As can be seen in the result statistics, there are a lot more capAdd calls done. For example, we see 384 capAdd function calls for cap_assign_type 2. That number can be split in 144 calls for caps to ground (GND) and 240 elemAdd function calls for couple caps. Only 6 of these calls result in a real new couple cap (call to function elemNew).
A number of 52 returns directly from the elemAdd function, because both nodes are the same. A number of 182 calls is used to change the existing couple cap element value. Thus, it is important that cap elements between nodes can fast be found. Function returnElement (see lump/elem.c) uses a hashing method to search for elements. I have not yet done execution time analysis of the *space3d* program for both cap_assign_type methods. See also appendix D and E for some other results.

Note that each calculated green value between two spiders is added as a ground cap. The value calculated for spiders (sp1,sp1) is added as one ground cap between sp1→subnode and subnGND. The value calculated for spiders (sp1,sp2) is added as two ground caps, one between sp1→subnode and subnGND and one between sp2→subnode and subnGND. In the new method sp1→subnode is changed into sp1→face→corners[0..3]. Thus, we get four* times more capAdd calls for ground caps (see figure below).



The value calculated for two different spiders (sp1,sp2) is also added as a couple cap between these two spiders. And, because each spider has four* corner points, we get 16 couple cap adds (elemAdd calls) for each pair of different spiders (see figure below).



* In our example, but it is 3 for triangular faces.

2.2 Resume cap_assign_type 0

See the following addCap code fragment:

```
addCap (s1, s2, green_value)
{
    if (s1 == s2)
        capAdd (s1 -> subnode, s1 -> subnode2, green_value); // GND cap
    else {
        if (s1 -> subnode != s2 -> subnode)
            capAdd (s1 -> subnode, s2 -> subnode, -green_value); // Couple

        capAdd (s1 -> subnode, s1 -> subnode2, green_value); // GND cap
        capAdd (s2 -> subnode, s2 -> subnode2, green_value); // GND cap
    }
}
```

The calculated green_value in the spider matrix gives always a GND capacitance. Note that spider→subnode2 points to this GND subnode, but this node can also be the substrate node by substrate calculation. As seen above, couple caps are only made for unequal subnodes. This gives in piecewise constant mode asymmetric assignment results, because too many center spiders point to the same face subnode. Note that the off diagonal green values are negative. To get positive couple caps a minus sign must be used. Note that the GND cap values are substracted from the diagonal GND cap values.

2.3 Resume cap_assign_type 1

See the following couple cap code fragment:

```
    if (s1 -> subnode != s2 -> subnode || cap_assign_type == 1
        && (s1 -> nom_x != s2 -> nom_x || s1 -> nom_y != s2 -> nom_y)) {
            v = sp4 ? -green_value / 4 : -green_value / 3;
            cap3dAdd (s1, sp1 -> subnode, v);
            cap3dAdd (s1, sp2 -> subnode, v);
            cap3dAdd (s1, sp3 -> subnode, v);
            if (sp4) cap3dAdd (s1, sp4 -> subnode, v);
        }
```

By cap_assign_type 1, couple caps are always done, except for equal subnodes where the spider center (x,y) coordinates are equal. Note that spiders sp1 .. sp4 are taken from s2→face→corners[0..3]. Function cap3dAdd does capAdd 4 times (or 3 times for triangular faces) for the corner spiders of center spider s1.

2.4 Resume cap_assign_type 2

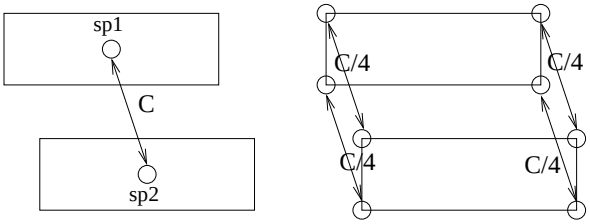
See the following couple cap code fragment:

```
    if (cap_assign_type == 2) {
        v = sp4 ? -green_value / 4 : -green_value / 3;
        cap3dAdd (s1, sp1 -> subnode, v);
        ...
    }
```

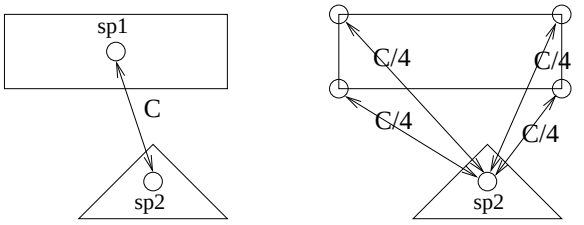
As can be seen, by cap_assign_type 2, the couple caps are always done. Note that this not always result in 16 couple caps for faces with 4 corners. Because in the z-direction, the lower and upper face of a conductor point to the same subnodes. Thus, 4 couple caps have the same subnodes. Note that also touching horizontal and vertical faces have 2 corner points in common.

2.5 Resume cap_assign_type 3

A more simple couple cap assignment is done for cap_assign_type 3. See the following figure:



Note that this gives no couple caps for lower / upper faces of the same conductor in the z-direction. In case the faces have a different number of corners the following assignment is done:



2.6 Resume cap_assign_type 4

The cap_assign_type 4 is like cap_assign_type 3, but shall use the more complicated assignment of cap_assign_type 2, in cases where the number of face corners is different. Thus, shall try to make 12 couple caps in that case.

3. APPENDICES

APPENDIX A -- Source File Changes

```
date: 2004/04/05 09:43; added substrate subnode to face (SdeG)
================================================================================
spider/refine.c

62a63
> extern bool_t prePass1;

face_t *meshRenameFace (sp1, sp2, fc)
{
    face_t *newfc = newFace (fc -> core, fc -> contact, fc -> coarse);
>   newfc -> sc_subn = fc -> sc_subn;
    ...
}

Private void splitTriangle (face, sp1, sp2, sp3, spN)
{
    ...
    newf = newFace (face -> core, face -> contact, face -> coarse);
>   newf -> sc_subn = face -> sc_subn;
    ...
}

Private void meshAddExtraSpider (face) face_t *face;
{
    ...
<   subnode_t * subnodeSC;
>   subnode_t *subnode, *subnodeSC;
    ...

#define MORE_XY(sp) \
        if (sp -> nom_x > sp_subnode -> nom_x \
        || (sp -> nom_x == sp_subnode -> nom_x \
         && sp -> nom_y > sp_subnode -> nom_y)) sp_subnode = sp;

<   sp_subnode = sp1;
<   MORE_XY (sp2);
<   MORE_XY (sp3);
<   if (sp4) MORE_XY (sp4);
<
<   if (subnodeSC = sp_subnode -> subnode2) {
<     if (!sp1 -> subnode2 || !sp2 -> subnode2
<      || !sp3 -> subnode2 || (sp4 && !sp4 -> subnode2)) subnodeSC = NULL;
<   }
---
>   if (prePass1) { /* optSubRes */
>       /* For substrate, the node cannot always correctly be found by
>          edge spiders (this can happen by triangles).
>          Thus the face is used to find the substrate contact subnode.
>       */
>       subnode = face -> sc_subn;
>       subnodeSC = NULL;
>   }
>   else {
>       sp_subnode = sp1;
>       MORE_XY (sp2);
>       MORE_XY (sp3);
>       if (sp4) MORE_XY (sp4);
>       subnode = sp_subnode -> subnode;
```

```
>       subnodeSC = face -> sc_subn;
>   }
>   ASSERT (subnode);

    spe = newSpider (nom_x, nom_y, nom_z, x, y, z, sp1 -> level,
<               sp_subnode -> subnode, subnodeSC, conductor, isGate);
>               subnode, subnodeSC, conductor, isGate);
    spe -> face = face;
    ...
}
================================================================================
spider/face.c

face_t * newFace (core, contact, coarse)
{
    ...
>   f -> sc_subn = NULL;
    ...
}
================================================================================
spider/sppair.c

43c43
< Private void tryFace P_((spider_t *sp1, spider_t *sp2, spider_t *sp3, spider_t *sp4,
> Private void tryFace P_((tile_t *t, spider_t *sp1, spider_t *sp2, spider_t *sp3, spider_t *sp4,
45c45
< Private void cosFace P_((spider_t *sp1, spider_t *sp2,
> Private void cosFace P_((tile_t *t, spider_t *sp1, spider_t *sp2,
52,54d51
< #if 0
< Private void spiderAdjustPosition P_((spider_t *sp, int orient, meshCoor_t offset));
< #endif

Private void doSpiderPair (tile_l, tile_r, bdr, orientation)
{
    ...
314c310
<               tryFace (b1, tl1[j], tl2[j], b2, m -> core_l, TRUE, coarse);
>               tryFace (tile_l, b1, tl1[j], tl2[j], b2, m -> core_l, TRUE, coarse);
328c324
<               tryFace (b1, b2, tr2[j], tr1[j], m -> core_r, TRUE, coarse);
>               tryFace (tile_r, b1, b2, tr2[j], tr1[j], m -> core_r, TRUE, coarse);
345c341
<               tryFace (b1, tl1[i], tl2[i], b2, m -> core_l, TRUE, coarse);
>               tryFace (tile_l, b1, tl1[i], tl2[i], b2, m -> core_l, TRUE, coarse);
359c355
<               tryFace (b1, b2, tr2[i], tr1[i], m -> core_r, TRUE, coarse);
>               tryFace (tile_r, b1, b2, tr2[i], tr1[i], m -> core_r, TRUE, coarse);
367,368c363,364
<           tryFace (tl1[i], bl1[i], bl2[i], tl2[i], core, FALSE, coarse);
<           tryFace (tr1[i], tr2[i], br2[i], br1[i], core, FALSE, coarse);
---
>           tryFace (tile_l, tl1[i], bl1[i], bl2[i], tl2[i], core, FALSE, coarse);
>           tryFace (tile_r, tr1[i], tr2[i], br2[i], br1[i], core, FALSE, coarse);
378,381c374,377
<           if (tl1[i] && bl1[i]) cosFace (tl1[i], bl1[i], core, FALSE, coarse);
<           if (tl2[i] && bl2[i]) cosFace (tl2[i], bl2[i], core, FALSE, coarse);
<           if (tr1[i] && br1[i]) cosFace (tr1[i], br1[i], core, FALSE, coarse);
<           if (tr2[i] && br2[i]) cosFace (tr2[i], br2[i], core, FALSE, coarse);
---
>           if (tl1[i] && bl1[i]) cosFace (tile_l, tl1[i], bl1[i], core, FALSE, coarse);
>           if (tl2[i] && bl2[i]) cosFace (tile_l, tl2[i], bl2[i], core, FALSE, coarse);
>           if (tr1[i] && br1[i]) cosFace (tile_r, tr1[i], br1[i], core, FALSE, coarse);
>           if (tr2[i] && br2[i]) cosFace (tile_r, tr2[i], br2[i], core, FALSE, coarse);
```

```
389c385
<               tryFace (tl1[i], tr1[i], tr2[i], tl2[i], core, FALSE, coarse);
>               tryFace (tile_l, tl1[i], tr1[i], tr2[i], tl2[i], core, FALSE, coarse);
391c387
<               tryFace (bl1[i], bl2[i], br2[i], br1[i], core, FALSE, coarse);
>               tryFace (tile_l, bl1[i], bl2[i], br2[i], br1[i], core, FALSE, coarse);
399c395
<               cosFace (tl1[i], bl1[i], core, FALSE, coarse);
>               cosFace (tile_l, tl1[i], bl1[i], core, FALSE, coarse);
    ...
}

< Private void tryFace (sp1, sp2, sp3, sp4, core, contact, coarse)
---
> Private void tryFace (tile, sp1, sp2, sp3, sp4, core, contact, coarse)
> tile_t * tile;
{
    ...
    if (n > 3) { /* n == 4 || n == 5 -> set faces */
        face_t * face = newFace (core, contact, coarse);
>       if (tile -> subcont) face -> sc_subn = tile -> subcont -> subn;
        ...
    }
}

< Private void cosFace (sp1, sp2, core, contact, coarse)
---
> Private void cosFace (tile, sp1, sp2, core, contact, coarse)
> tile_t * tile;
{
    ...
    if (n == 4 || n == 3) {
        face_t * face = newFace (core, contact, coarse);
>       if (tile -> subcont) face -> sc_subn = tile -> subcont -> subn;
        ...
    }
}

Private spider_t * spiderNew (x, y, z, level, conductor, tile, isGate)
{
    ...
    if (prePass1) { /* optSubRes */
        ASSERT (tile -> subcont);
<       if (!(subnode = tile -> subcont -> subn))
<           subnode = tile -> subcont -> subcontInfo -> subn;
---
>       subnode = tile -> subcont -> subn;
    }
    else {
        if (!optRes)
            subnode = tile -> cons[conductor];
        else {
            ...
            closest_np = tile -> rbPoints;
            ...
            np = closest_np -> next;
            while (np) { ... }
            np = tile -> tlPoints;
            while (np) { ... }
            subnode = closest_np -> cons[conductor];
        }

        if (tile -> subcont) {
<           if (!(subnodeSC = tile -> subcont -> subn))
```

```
<               subnodeSC = tile -> subcont -> subcontInfo -> subn;
---
>           subnodeSC = tile -> subcont -> subn;
>           ASSERT (subnodeSC);
        }
    }
    ASSERT (subnode);
    ...
}

< #if 0
<    ...
< Private void spiderAdjustPosition (spider, orientation, offset)
<    ...
< #endif

Private face_t * spiderFindFace (tile, level, core, coarse)
{
>   face_t *f;

<   if (!tile -> mesh -> faces[level])
<       tile -> mesh -> faces[level] = newFace (core, FALSE, coarse);
---
>   if (!(f = tile -> mesh -> faces[level])) {
>       tile -> mesh -> faces[level] = f = newFace (core, FALSE, coarse);
>       if (tile -> subcont) f -> sc_subn = tile -> subcont -> subn;
>   }

<   return (tile -> mesh -> faces[level]);
---
>   return (f);
}
================================================================================
spider/triang.c

Private void mkTriangle (sp1, sp2, sp3, face, doRecur)
{
    face_t * newface = newFace (face -> core, face -> contact, face -> coarse);
>   newface -> sc_subn = face -> sc_subn;
    ...
}
================================================================================
include/spider.h

typedef struct Face {
    ...
>   struct subnode * sc_subn;        /* substrate contact subnode */
    ...
} face_t;
================================================================================
```

APPENDIX B -- Extraction Results of Network t0

Layout of cell t0:

```
:: lambda = 0.001 micron
ms t0
box  m1  0 12  0 28
term m1  0  0  0  0 A
term m1  0  0 28 28 B
term m1 12 12  0  0 C
term m1 12 12 28 28 D
me
```

Technology definitions:

```
conductors:    met1 : m1 : m1 : 63e-3
capacitances:  cap1 : m1 : m1 : 22.6e-3         # for -C
vdimensions:   dim1 : m1 : m1 : 1.89e-6 0.35e-6 # for -3C
dielectrics:   SiO2   2.65  0.0                 # for -3C
```

Parameters for space3d:

```
cap3d.max_be_area   10
cap3d.be_window     10
low_sheet_res     1e-3
```

SLS networks:

```
network t0 (terminal A, B, C, D) /* -rC */
{
    res 294m (B, A);
    res 54m (B, D);
    cap 1.8984e-18 (B, GND);
    res 54m (A, C);
    cap 1.8984e-18 (A, GND);
    res 294m (C, D);
    cap 1.8984e-18 (C, GND);
    cap 1.8984e-18 (D, GND);
}
network t0 (terminal A, B, C, D) /* -r3C */
{
    res 294m (B, A);
    res 54m (B, D);
    cap 10.00939e-18 (B, C);
    cap 23.87118e-18 (B, D);
    cap 3.733122e-18 (B, GND);
    res 54m (A, C);
    res 294m (C, D);
    cap 8.534915e-18 (C, D);
    cap 3.410486e-18 (C, GND);
    cap 9.02205e-18 (D, GND);
}
```

Spider green matrix:

```
  | 1 (D)       | 2 (D)       | 3 (D)       | 4 (C)       | 5 (D)       | 6 (B)       |
--+-------------+-------------+-------------+-------------+-------------+-------------+
1 | 3.76137e-17 |-2.73334e-19 |-1.00094e-17 |-1.00094e-17 |-2.72907e-19 |-1.33155e-17 |
2 |             | 1.65184e-18 |-8.87279e-20 |-8.87279e-20 | 1.05746e-20 |-2.73334e-19 |
3 |             |             | 2.19548e-17 | 1.65154e-18 |-8.83374e-20 |-1.00094e-17 |
4 |             |             |             | 2.19548e-17 |-8.83374e-20 |-1.00094e-17 |
5 |             |             |             |             | 1.65206e-18 |-2.72907e-19 |
6 |             |             |             |             |             | 3.76137e-17 |
```

APPENDIX C -- New Extraction Results of Network t0

Technology definitions:

```
conductors:    met1 : m1 : m1 : 63e-3
vdimensions:   dim1 : m1 : m1 : 1.89e-6 0.35e-6 # for -3C
dielectrics:   SiO2   2.65  0.0                 # for -3C
```

Parameters for space3d:

```
cap3d.cap_assign_type 1
cap3d.max_be_area   10
cap3d.be_window     10
low_sheet_res     1e-3
```

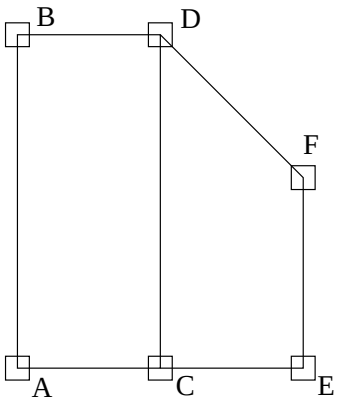
SLS networks:

```
network t0 (terminal A, B, C, D) /* -r3C : cap_assign_type 1 */
{
    res 294m (B, A);
    res 54m (B, D);
    cap 4.772636e-18 (B, A);
    cap 8.514408e-18 (B, D);
    cap 8.101523e-18 (B, C);
    cap 4.041414e-18 (B, GND);
    res 54m (A, C);
    cap 8.101523e-18 (A, D);
    cap 8.514408e-18 (A, C);
    cap 4.041414e-18 (A, GND);
    res 294m (C, D);
    cap 4.772636e-18 (C, D);
    cap 4.041414e-18 (C, GND);
    cap 4.041414e-18 (D, GND);
}
network t0 (terminal A, B, C, D) /* -r3C : cap_assign_type 2 */
{
    res 294m (B, A);
    res 54m (B, D);
    cap 4.771314e-18 (B, A);
    cap 8.513086e-18 (B, D);
    cap 8.100201e-18 (B, C);
    cap 4.041414e-18 (B, GND);
    res 54m (A, C);
    cap 8.100201e-18 (A, D);
    cap 8.513086e-18 (A, C);
    cap 4.041414e-18 (A, GND);
    res 294m (C, D);
    cap 4.771314e-18 (C, D);
    cap 4.041414e-18 (C, GND);
    cap 4.041414e-18 (D, GND);
}
```

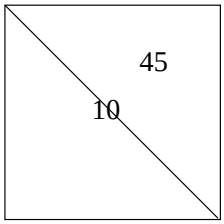
Program statistics:

cap_assign_type	0	1	2	3	
addCap	21	21	21	21	
capAdd	45	368	384	204	
== GND	36	144	144	144	# GND caps
elemAdd	9	224	240	60	
elemNew	3	6	6	6	# Couple caps
elemChange	6	170	182	46	
nA==nB	0	48	52	8	

APPENDIX D -- Extraction Results of Network t1



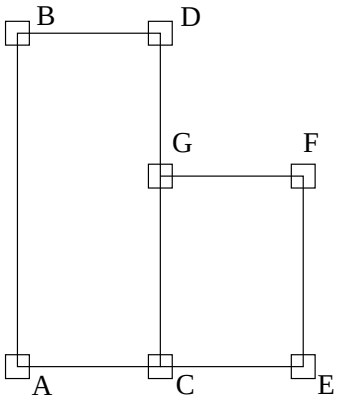
6 nodes → 15 couple caps
10 spiders → 55 addCap's



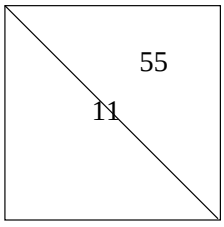
10 x 10

cap_assign_type	0	2	3	
capAdd	136	1120	580	
== GND	100	400	400	# GND caps
elemAdd	36	720	180	(720 = 45 x 16)
elemNew	10*	15	15	# Couple caps
elemChange	26	601	147	*) to A missing
nA==nB	0	104	18	

APPENDIX E -- Extraction Results of Network t2



7 nodes → 21 couple caps
11 spiders → 66 addCap's



11 x 11

cap_assign_type	0	2	3	
capAdd	164	1364	704	
== GND	121	484	484	# GND caps
elemAdd	43	880	220	(880 = 55 x 16)
elemNew	10*	21	21	# Couple caps
elemChange	33	755	181	*) to A,G missing
nA==nB	0	104	18	