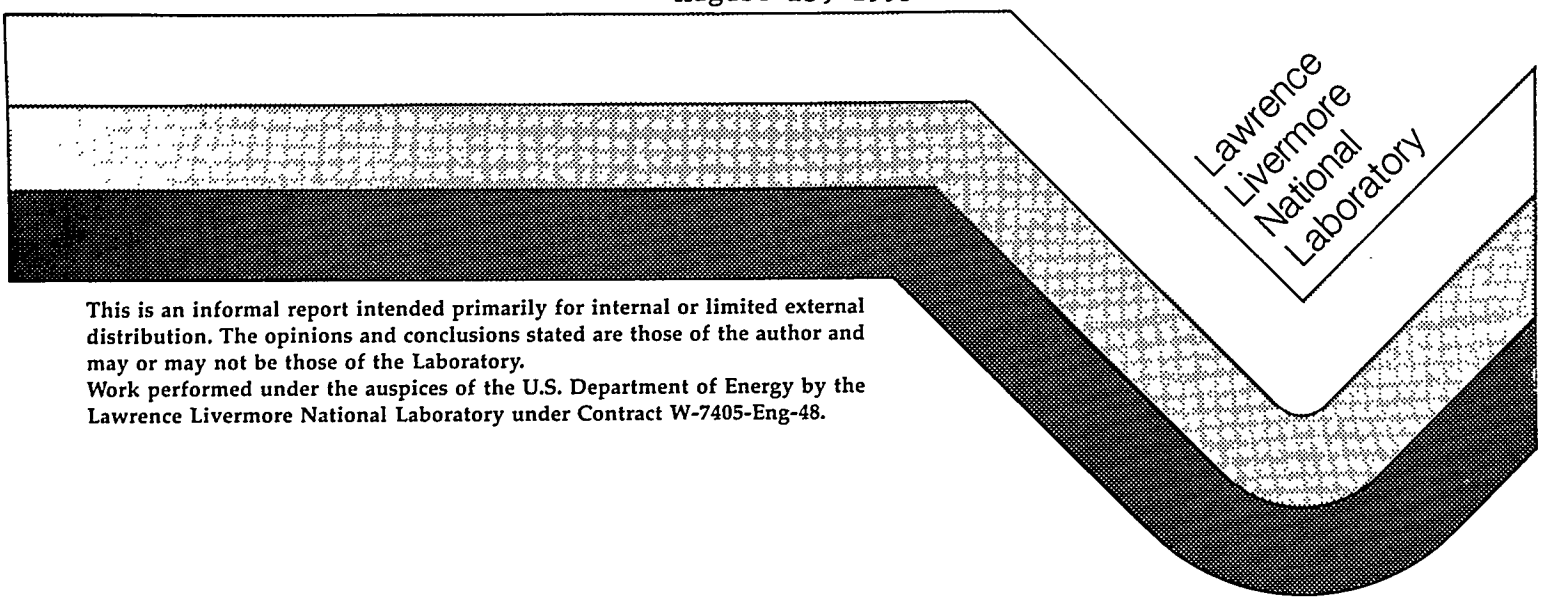
UCRL-ID- 121835

DSI3D - RCS
User Manual

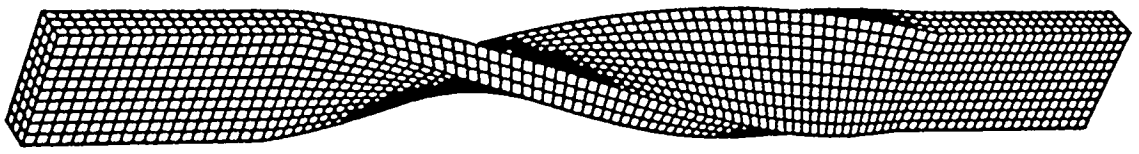
Niel Madsen
David Steich
Grant Cook
Bill Eme

August 23, 1995



DSI3D - RCS

User Manual



August 23, 1995

Niel Madsen, David Steich, Grant Cook, Bill Eme

Lawrence Livermore National Laboratory

Table of Contents

Introduction

The **DSI3D-RCS** code is designed to numerically evaluate radar cross sections on complex objects by solving Maxwell's curl equations in the time-domain and in three space dimensions. The code has been designed to run on the new parallel processing computers as well as on conventional serial computers.

The **DSI3D-RCS** code is unique for the following reasons:

- Allows the use of unstructured non-orthogonal grids,
- Allows a variety of cell or element types,
- Reduces to be the Finite Difference Time Domain (FDTD) method when orthogonal grids are used,
- Preserves charge or divergence locally (and globally),
- Is conditionally stable,
- Is selectively non-dissipative,
- Is accurate for non-orthogonal grids.

This method is derived using a Discrete Surface Integration (DSI) technique. As formulated, the DSI technique can be used with essentially arbitrary unstructured grids composed of convex polyhedral cells. This implementation of the DSI algorithm allows the use of unstructured grids that are composed of combinations of non-orthogonal hexahedrons, tetrahedrons, triangular prisms and pyramids. This algorithm reduces to the conventional FDTD method when applied on a structured orthogonal hexahedral grid.

Installation

DSI3D has been designed to run on message passing MIMD parallel computers such as the Intel Paragon, the Intel iPSC/860 and the Meiko CS-2. It uses MPI or the Intel NX message passing library routines for performing inter-processor communication. Almost all of the code is written in standard Fortran 77 augmented with calls to the message passing routines. A small portion is written in standard C which provides for dynamic array allocation. The code also uses Cray style pointers.

DSI3D also uses numerous binary and ascii disk files which can be located either on a global parallel disk system or on local disks associated with each processor (such as on the Meiko CS-2).

A tar file is provided which contains all of the required **DSI3D** related codes, makefiles and sample problems. A user should consult the **README** file contained within the tar file for specific installation requirements.

The basic stages in solving a problem using **DSI3D** consist of the following:

- Mesh Generation: **GRIDTOBIN**
- Mesh Partitioning: **GRIDPART, HORST** and **PARTITIONTOBIN**
- Input Preparation: **params.dat** file
- Execution of the geometry assembling pre-processor: **PREDSI3D**
- Execution of the time stepping code: **DSI3D**
- Evaluation of RCS values: **RCSS**
- Preparation of visualization data: **DSI3DIO**

Mesh Generation

An ascii mesh file is required to be generated for each problem. There are many mesh generator codes available from numerous vendors. They all work differently but in essence they all take geometry information as supplied by the user or a CAD package and use this information to generate nodes, elements and surfaces. **DSI3D** requires the data to be in the following format (a standard format when using the mesh generators TrueGrid or Ingrid):

Line 1: Title line: (a80)

Line 2: #materials, #nodes, #elements, #surfaces: (4i8)

Next #nodes lines: node number, x-coordinate, y-coordinate, z-coordinate (i8, 3e18.9)

Next #elements lines: element number, material number, node numbers (8), (i8,i5,8i8)

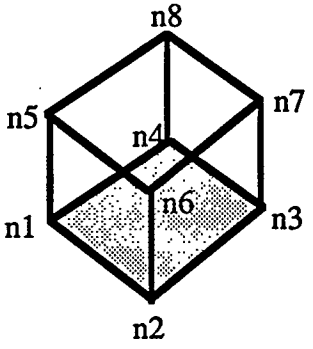
Next #surfaces lines: ascii type, surface type id, node numbers (4), (a3,i5,4i8)

The node numbers, element numbers and ascii surface types are not actually used by **DSI3D** but are there for user convenience in editing, searching and modifying an existing mesh file.

The material number identifies a 4 parameter set of material constants subsequently specified in the input data. They are permittivity, permeability, electric conductivity and magnetic conductivity.

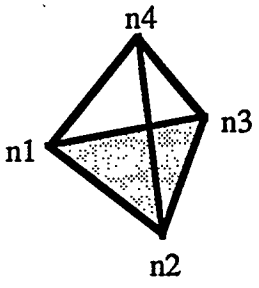
The elements (or grid cells) are defined by 8 node numbers specified in a particular order. Four different elements are allowed to coexist in the same grid: hexahedra, tetrahedra, triangular prisms and pyramids. If n nodes are required to define an element then the remaining 8-n node numbers should be set to zero.

Hexahedrons are defined by 8 nodes, the first 4 nodes define the base and the next 4 nodes define the top. When viewed from a position inside the element, the base nodes must be defined in a counterclockwise direction as pictured below.



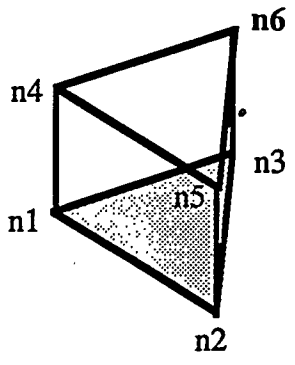
Hexahedron node ordering (base shaded).

Tetrahedra are define by 4 nodes, the first 3 nodes define the base and the last node defines the apex (relative to the base). When viewed from a position inside the element, the base nodes must be defined in a counterclockwise direction as pictured below.



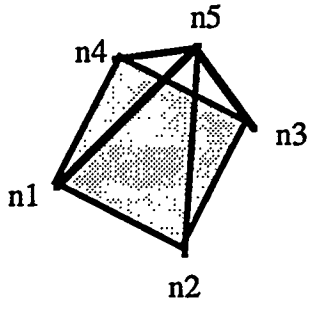
Tetrahedron node ordering (base shaded).

Triangular prism elements are defined by 6 nodes, the first three define the base and the last three the top as pictured below. When viewed from a position inside the element, the base nodes must be defined in a counterclockwise direction as pictured below.



Triangular prism node ordering (base shaded).

Pyramid elements are define by 5 nodes, the first 4 define the base and the last node defines the apex (relative to the base) as pictured below. When viewed from a position inside the element, the base nodes must be defined in a counterclockwise direction as pictured below.



Pyramid element node ordering (base shaded).

The surface definitions are used to define boundary conditions, far-field integration surfaces, incident field surface, etc. The ascii identification field is provided for user convenience only. **DSI3D** uses only the integer surface type identifier and the 4 nodes which define the surface patch. If the surface patch is triangular, then only 3 nodes are required and the fourth node should be set to zero.

Special surfaces are of surface type=1 (typically driven PEC surfaces).

Surface type 50 is reserved for the far field integration surface. At present, all surface patches must directly "see" the the far field reference point defined in the parameter file without haveing to pass through another part of the far field integration surface. In other words, the integration surface must be a simple surface that does not fold on itself (i.e., a starred surface).

Valid RBC surface types range from 51-69. Each corresponding surface type used in the mesh file should be defined in the setup parameter file. If no definition is found in the setup parameter file then a PEC boundary condition is assumed.

There are presently two restrictions that are enforced on the grid near the boundary condition surface.

1) The outer surface edges where the RBC is to be applied must coincide with two other interior edges that are oriented in parallel with the RBC edge.
2) The midpoints of the RBC edge and the two interior edges must be collinear and equidistant.

As a consequence of the above restrictions and the finite number of primary element shapes supported, the outer two layers of the mesh must be constructed of hexahedral or triangular prism elements. If triangular prism elements are used, a triangular face must exist on the outer surface of the mesh with triangular faces adjoining other triangular faces. The lengths of the edges may be different and there are no restrictions placed on adjoining cell shapes and sizes. Although the above restrictions are strict, the outer RBC surface can be almost any shape. For best RBC performance it is suggested that the inward directed normal of the outer RBC surface be directed toward the primary scattering center and be spatially located as far from the scatterer as is practical.

Grid checks are performed to enforce the two above restrictions. Often grid generators may not produce exact node placements to satisfy the above restrictions within some small tolerances. If the grid restrictions fail then the user will be notified via an error message. The user may relax some of these tolerances by editing the source code. This is not recommended but may be the only option available if the grid generator fails to produce a satisfactory grid. See subroutine: "ck_dir" in orbc.f for directions on changing the orthogonality tolerance.

Once the ascii mesh file has been produced, the code **GRIDTOBIN** should be executed in order to convert the ascii mesh file into two binary files: **nodesb.dat** and **elementsb.dat.** The file **nodesb.dat** contains the node coordinates and the file elementsb.dat contains the element and surface definititions. This conversion to a binary format is done so that more rapid reading of the mesh file can occur. **GRIDTOBIN** executes only on serial computers. It requires the ascii mesh file and name as input, and produces the above mentioned files as output. An error will result if either of the two binary files exists before running **GRIDTOBIN.**

Mesh Partitioning

The next step in input preparation is to partition the mesh so that sub-pieces can be used by the different processors. The codes **GRIDPART** and **HORST** are used for this purpose. **GRIDPART** uses the previously defined ascii mesh file to produce a connectivity graph in a special sparse format which is then read by the **HORST** code to actually perform the partitioning.

HORST produces an ascii partition file which contains exactly one integer per line for each element in the mesh. These integers range from 0 to #processors - 1. If the i-th entry of the partition file is j, then element i is assigned to processor j for processing.

PARTITIONTOBIN is next executed. It reads the ascii partition file and produces an equivalent binary partition file named: **assgnelb.dat** .

If **DSI3D** is to be run only on a serial computer, then these three codes need not be executed at all as the information they produce is irrelevant.

GRIDPART, HORST and **PARTITIONTOBIN** are run only on serial computers and request the user to supply a partition factor and the ascii mesh file name. The partition factor must be an integer and is $\log_2$(#processors). For example if 8 processors are to be used, then the partition factor is 3. Very large problems may exceed the capability of of many serial computers and in such cases alternative ways of partitioning the problem must be found. For example, a two-dimensional cross section of the grid (hopefully much smaller than the entire grid) may be partitioned and then generalized or extended to the full three-dimensional grid.

HORST is an implementation of the Recursive Spectral Bisection method developed by Horst Simon[2]. Other methods such as recursive coordinate bisection have been proposed as alternatives to the **HORST** approach.

Running PREDSI3D

PREDSI3D is a preprocessor which assembles all of the geometry information from the mesh files and input decks and assembles it into new files which provide all of the input information that **DSI3D** requires to execute. It computes array size information, problem control parameters, coefficients and pointers required to update any given edge data, as well as far field data and interprocessor communication data.

The execution of **PREDSI3D** requires an input deck with the name **params.dat**. For a description of the commands available see the **Input Setup** section. Files that are required for input and created for output, exist in either global or local file space. The definition of local and global space is defined in the file **params.dat**.

PREDSI3D has been designed to run on a parallel computer, but it can be run on a single processor machine if the problem is sufficiently small.

Input Files:

params.dat	File contains the commands used to drive execution of the problem. This file is required, and is expected to exist in the current working directory.
nodesb.dat	Binary file containing the node definitions of the mesh. This file is created by running the executable **GRIDTOBIN** on the ascii input mesh file. This file is required and should exist in the global file space.
elementsb.dat	Binary file containing the element and surface definitions of the mesh. This file is created by running the executable **GRIDTOBIN** on the ascii input mesh file. This file is required and should exist in the global file space.
assgnelb.dat	Binary file containing the processor number for each element in the mesh. This file is created by running the executable **PARTITIONTOBIN** on the ascii partition file. This file is optional and should exist in the global file space. If the file does not exist, it is assumed that the problem is running in single processor mode.

Output Files:

rundatXXXX	Ascii files containing problem size information and other various problem control information for the processor XXXX. These files are created in global space.

nptrXXXX Binary files containing encoded material information and edge numbers used for the update equations for both H and E edges. These files may also contain edge information for far field and RBCs if these options are enabled. These files are created in local file space for each processor.

coefsXXXX Binary files containing coefficient values for each contributing edge for the update equations. These files may also contain edge information for far field and RBCs if these options are enabled. These files are created in local file space for each processor.

ffieldXXXX Binary files containing far field specific information. If the far field option was not specified these files will not exist. These files are created in local file space for each processor.

commXXXX Binary files containing edge values to communicate with other processors after each update equation. If the problem is run in single processor mode, these files will not exist. These files are created in local file space for each processor.

Running DSI3D

The execution of **DSI3D** requires an input deck with the name **params.dat**. This input deck should be identical to the input deck provided to **PREDSI3D**. The only options used from the input deck are **local_dir, global_dir,** and **msg_lib**. Files that are required for input and created for output, exist in either global or local file space. The definition of local and global space is defined by the file **params.dat**.

DSI3D has been designed to run on a parallel computer, but it can be run on a single processor machine if the problem is sufficiently small.

Input Files:

params.dat	File contains the commands used to describe the locations of the files. This file is required, and is expected to exist in the current working directory.
rundatXXXX	Ascii files containing problem size information and other various problem control information for the processor XXXX. These files are located in global space.
nptrXXXX	Binary files containing encoded material information and edge numbers used for the update equations for both H and E edges. These files may also contain edge information for far field and RBCs if these options are enabled. These files are located in local file space for each processor.
coefsXXXX	Binary files containing coefficient values for each contributing edge for the update equations. These files may also contain edge information for far field and RBCs if these options are enabled. These files are located in local file space for each processor.
ffieldXXXX	Binary files containing far field specific information. If the far field option was not specified these files will not exist. These files are located in local file space for each processor.
commXXXX	Binary files containing edge values to communicate with other processors after each update equation. If the problem is run in single processor mode, these files will not exist. These files are located in local file space for each processor.

Output Files:

restartYYY	An ascii file in global file space containing the last time step number since last restart dump.The **YYY** is the total number of processors running the problem.
rXXXNNNNN	A binary file in local file space containing all of the information necessary for that processor to restart a problem from that time step. The **XXX** is the processor number, and **NNNNN** is the time step number of the restart.
tpYYYNNNNN	Ascii file containing time history files of user requested primary edge values. These files are created in the global file space. The **YYY** is the total number of processors running the problem, and **NNNNN** is the the beginning cycle number contained in the file. A new file is created upon every restart dump.
tdYYYNNNNN	Ascii file containing time history files of user requested dual edge values. These files are created in the global file space. The **YYY** is the total number of processors running the problem, and **NNNNN** is the the beginning cycle number contained in the file. A new file is created upon every restart dump.
ufar_outXXXX **wfar_outXXXX**	Binary files containing information about the far field. These files are created in the local file space. The **XXXX** is the processor number. If the far field option is not specified, these files will not exist.
rcss.out	Binary file containing the global information about the far field. This file is created in the global file space. If the far field option is not specified, these files will not exist.
newheXXXX	Binary file containing the map of the original edge locations to their new position. This file is created in the local file space with **XXXX** is the processor number.

Radar Cross Section Evaluation

The execution of **RCSS** is used to evaluate the radar cross sections of the simulation. It evaluates the far fields from the computed far field vector potentials and produces a file which can then be plotted to obtain the desired RCS information.

RCSS requires an input deck with the name **params.dat**. This input deck should be identical to the input deck provided to **PREDSI3D**. The only options used from the input deck are **local_dir, global_dir,** and **msg_lib**. Files that are required for input and created for output, exist in either global or local file space. The definition of local and global space is defined by the file **params.dat**.

RCSS has been designed to run on a parallel computer, but it can be run on a single processor machine if the problem is sufficiently small.

Input Files:

params.dat	File contains the commands used to describe the locations of the files. This file is required, and is expected to exist in the current working directory.
ufar_outXXXX **wfar_outXXXX**	Binary files containing information about the far field. These files are created in the local file space. The XXXX is the processor number.
rcss.out	Binary file containing the global information about the far field. This file is located in the global file space.

Output Files:

ultra.out	An ascii file containing the RCS information in a plottable format. The information contained is given in "time value" pairs, one pair on each line. For each far field point, E_θ, E_ϕ, H_θ, H_ϕ, and σ_1 (cross section polarization) are given, each as separate output histories. If a second polarization is specified in **params.dat**, then the σ for that polarization is also included.

Post-Processing for Visualization

DSI3DIO is the post processing code used for producing ascii data for visualization. It extracts specified edge values from either the electric, magnetic, or Poynting vector fields, and interpolates them to the nodes of the mesh. The extraction process produces vector fields either in planar slices or at material interfaces.

It is expected that for most problems that the entire data set will be too large to visualize on conventional workstations and that visualization from the parallel computer is not practical. The philosophy then is to have the parallel computer extract specified data subsets of the global data from the individual processors. This data is then placed into a single grid and data file pair which can then be easily transported from the parallel computer to a visualization workstation.

DSI3DIO requires an input deck with the name **params.dat** and the graphics command file **ioinput**. The **params.dat** file should be identical to the input file provided to **PREDSI3D**. The only options used from this file are **local_dir, global_dir,** and **msg_lib**. Files that are required for input and created for output, exist in either global or local file space. The definition of local and global space is defined by the file **params.dat.**

DSI3DIO has been designed to run on a parallel computer, but it can be run on a single processor machine if the problem is sufficiently small.

Input Files:

params.dat	Ascii file containing the commands used to describe the locations of the files. This file is required and is expected to exist in the current working directory.
ioinput	Ascii file containing commands to describe the data to extract from the problem. This file is located in global space.
rXXXNNNNN	A binary restart file located in local file space containing all of the edge values that will be interepeted to the nodes. The **XXX** is the processor number, and **NNNNN** is the time step number of the restart.
newheXXXX	Binary file containing the map of the original edge locations to their new position. This file is located in local file space. The **XXXX** is the processor number.

Output Files:

PRE_ENNNN **PRE_HNNNN** **PRE_PNNNN**	Ascii files located in global file space containing the X, Y, and Z component values at the nodes of the vector fields that were requested by the user for a given timestep. The **PRE_** is the prefix specified in the **ioinput** file, and **NNNNN** is the time step number of the data.
GRIDNAME	Ascii file reduced grid file of the data requested by the user. **GRIDNAME** is specified by the user in the **ioinput** file.

Form of DSI3DIO Input File

The form of the **ioinput** file is as follows:

nmat	The number of materials for which we are interested in extracting the vector field data
mat1	The first material number for which data is to be extracted.
mat2	The second material number for which data is to be extracted.
.	
.	
.	
matn	The Nth or last material number for which data is to be extracted.
planar_cut	Do we want the data to lie on a planar cut(s) (y/n)
xp1 yp1 zp1	For the first planar cut: the coordinates of a point on the plane.
xn1 yn1 zn1	For the first planar cut: a vector normal to the plane.
tol1	For the first planar cut: the floating point tolerance used to test whether or not a point lies on the plane.
.	
.	
.	
xpM ypM zpM	For the Mth or last planar cut: the coordinates of a point on the plane.
xnM ynM znM	For the Mth or last planar cut: a vector normal to the plane.
-tolm	For the Mth or last planar cut: the floating point tolerance used to test whether or not a point lies on the plane. Note that the last entry for this data section is flagged by setting the tolerance to a negative value.
reduced_grid	Do we want a reduced grid file (y/n). Note: this is required unless you already have produced the proper reduced grid from a previous **DSI3DIO** extraction.
gridname	The name of the reduced gridfile.
field_types	The type of vector field values to extract (e=electric, h=magnetic, p=Poynting). Any combination of length 1-3 of the ascii characters e, h, and p is allowed.
prefix	The prefix to pre-append on all field output file names

start_step stop_step step_increment
The beginning time step number, the ending time step number, and the time step increment between data extractions.These numbers must match the restart dump file time step numbers.

EXAMPLE Input

The following file extracts the node values that reside at all of the material interfaces of materials 3 and 1 for both the electric and magnetic fields. It produces a grid file of the name reduced.grd.

```
2
3
1
n
y
```

reduced.grd
eh
mat13_
100 2000 100

Input Setup

The commands available in **PREDSI3D** and **DSI3D** can be broken down into the following categories:

- **Problem Control Commands**
- **Excitation Sources Commands**
- **Material Commands**
- **Far Field Commands**
- **RBC Commands**
- **Output Commands**

Problem Control Commands

The problem control commands are used to determine how long the problem should be run, where the files should be created, and what type of message passing should be used. The commands are: **start_time, stop_time, num_steps, dt, restarts, local_dir, global_dir,** and **msg_lib.**

start_time *val*

The initial time to start the simulation. This command is required to exist in the input deck.

stop_time *val*

The time the simulation is to complete. This command is optional, and has a default value of **-1.e+20.** If this command is not present in the input deck, it is expected that the **num_steps** command is specified.

num_steps *ival*

The number of time steps to execute the simulation. This command is optional, and has a default value of **0.** If this command is not present in the input deck, it is expected that the **stop_time** command is specified.

dt *val*

The time step to take between each iteration of the simulation. This command is optional, and has a default value of **0.0.** If this command is not present in the input deck, the time step is determined by taking 0.5 the time it takes the speed of light to travel across the minimum edge length in the problem.

restarts *ival*

The frequency in time steps to write out restart dumps. This command is required to exist in the input deck.

local_dir *sval*

Pathname describing the location of where to create local files. This command is optional and has a default value of **/tmp/.**

global_dir *sval*

Pathname describing the location of where global files are located This command is optional and has a default value of **./.**

msg_lib *sval*

The type of message passing library to use. This command is optional with a default value of **mpsc.** Currently the only other acceptable value is **mpi.** If running the code in scalar mode, it is suggested to use the value **mpsc.**

alpha **val**

The dissipation control parameter. It has a default value of 1.0e+06. Any value of **alpha** greater than 1.0e+05 will cause **DSI3D** to run in a non-dissipative mode (the preferred mode). For increased stability, a value of 30.0 is suggested. For grids with extreme distortion, smaller values (more dissipation) of **alpha** may be necessary. Values of **alpha** smaller than 5.0 to 6.0 are **not** recommended.

Excitation Sources Commands

The excitation sources commands are used to control the amount and the method of adding energy into a problem. The commands are: **driver_id, driver,** and **current_edges.**

driver_id *ival*

This value is used to define the type of incident field to use for driven surfaces. Currently the only valid value is 7. Use of any other value will cause indeterminate results.

driver *thinc phinc polang amp delay beta dum tmin tmax dum*

This command defines the values for various driver parameters. **driver_id=7** is a Gaussian pulse. The meaning for each argument for **driver_id=7** is:

thinc The theta incident angle.
phinc The phi incident angle.

These two parameters give the direction (θ, ϕ) that points to the source of the incident pulse. The spherical coordinate system has θ=0 aligned with the positive z-axis and ϕ=0 aligned with the positive x-axis.

polang The polarization of the incident E-field. Zero degrees corresponds to a vertical polarization and 90 degrees corresponds to a horizontal polarization.
amp The amplitude of the pulse
delay The distance that the pulse needs to be offset in order not to overlap the scatterer at the initial time of the simulation. This can be a positive or negative number (or even 0), depending upon whether the pulse needs to be shifted toward the scatterer (positive) or away from the scatterer (negative). More precisely, the distance shifted should equal the largest value of $\mathbf{r}\cdot\mathbf{R}$, where $\mathbf{r}$ is any point on the scatterer, and $\mathbf{R}$ is a unit vector pointing to the source of the incident pulse, i.e., in the direction (thinc, phinc).
beta The beta of the pulse is the number of time steps it takes the pulse to e-fold 16 times. It is also the number of time steps from the start of the pulse to its peak.
dum Unused.
tmin The start time of the pulse (should always be **0.0**).
tmax The stop time of the pulse (should always be **2.0**).
dum Unused.

current_edges *node1 node2 [icnt1 inc1 icnt2 inc2]*

This specifies the edges to be used as current sources. The edges are specified by giving global node numbers defining the edge with an optional set of arguments which define a double do-loop structure for selecting multiple edges. The optional argument set will define the following set of node numbers.

```
node10 = node1
node20 = node2
do j=1,icnt2+1
        do i=1,icnt1+1
                node1 = node10 + (j-1)*inc2 + (i-1)*inc1
                node2 = node20 + (j-1)*inc2 + (i-1)*inc1
        enddo
enddo
```

Material Commands

The material commands are used to control the properties of the materials. The commands are: **eps0, xms0,** and **materials.**

eps0 *val*

This value sets the reference permittivity value for the entire problem. Its value must be non-zero.

xmu0 *val*

This value sets the reference permeability value for the entire problems. Its value must be non-zero.

materials ***matnum permittivity permeability sigmae sigmah***

matnum	material identification number as found in the mesh file
permittivity	material relative permittivity (relative to **eps0**)
permeability	material relative permeability (relative to **xmu0**)
sigmae	material electric conductivity
sigmah	material magnetic conductivity

Far Field Commands

The far field commands are used to control the far fields that are computed. The commands are: **fardim, farref, farpol,** and **farpts.**

fardim *ival*

A value used for specifying the dimensionality of a problem: 2=2D or 3=3D problem.

farref *val val val*

A point that is guaranteed to be inside of the scatterer.

farpol *val*

The polarization of the radar detector (zero degrees corresponds to vertical polarization).

farpts *theta_angle phi_angle*

Each pair of values specifies the direction of a far field point. Any number of such pairs may be given, each on a separate line.

RBC Commands

The **rbc** command is used to control the type of radiation boundary condition to use. The command is: **rbc**

rbc rbc_type tag-number <angle_null1> <loss1> <angle_null2> <loss2>

< > denotes an optional argument

rbc_type The actual type of rbc to be applied whenever a special surface with the given tag-number is encountered. Each type can also be adjusted to have different nulling angles and loss terms. Presently the following rbc types are supported:

higdon	nr	1st order Higdon
taylor	nr	1st order Taylor series
higdon_higdon	nr*	2nd order Higdon
higdon_taylor	nr*	1st order Higdon / 1st order Taylor
taylor_higdon	nr*	1st order Taylor / 1st order Higdon
taylor_taylor	nr*	2nd order Taylor
mur	n*	2nd order Mur, modified: 1½-D stencil
liao	nr	1st order Liao

* denotes a 2nd order boundary condition
n denotes allows user to define arbitrary nulling angle(s)
r denotes allows user to define arbitrary loss term(s)

tag-number Valid range is between 51-69 inclusive. Each different special surface tag-number found in the mesh file must have an rbc declaration line in this file. Missing declarations default to PEC.

angle_null1 Angle in degrees (off normal from rbc edge) at which the rbc is matched (i.e., angle of no reflection). One angle is allowed per order of rbc except for Mur. Mur can have only 1 null angle even though the rbc is second order overall. Valid range is 0-90 degrees. Suggested range is 0-60 degrees to help insure stability and to not degrade near normal incidence reflection.

loss1 Loss term. Valid for all boundary conditions but 2nd order Mur. Adding loss can help to maintain stability of the rbc. Adding too much loss can degrade the rbc performance. Valid range is 0.0 - 0.05. Suggested range is 0.0 - 0.02.

angle_null2 Same as **angle_null1** but for the second order part of the rbc. This argument is ignored if only a first order rbc is specified for the **rbc_type** above.

loss2 Same as **loss1** but for the second order part of the rbc. This argument is ignored if only a first order rbc is specified in the **rbc_type**.

Although any rbc can be specified over any special surface, ambiguity can arise as to which rbc to apply to a given rbc edge. This can occur when specifying different rbc's

over two special faces which share a common edge. This issue is unresolved. However, no error will result and the user is assured that one of the two rbc specifications will be applied to the common edge. Typically a user specifies only a single boundary condition over the surface and the above mentioned concern is irrelevant.

Output Commands

The output commands are used to control the near field edge values to get a time history of. The commands are: **dual_output_edges** and **primary_output_edges.**

dual_output_edges *ielem1 ielem2 [icnt1 inc1 icnt2 inc2]*

This specifies the dual edge values to save. The edges are specied by giving global element numbers defining the edge with an optional set of arguments which function as those given above for the command **current_edges.**

primary_output_edges *node1 node2 [icnt1 inc1 icnt2 inc2]*

This specifies the primary edge values to save. The edges are specied by giving global node numbers defining the edge with an optional set of arguments which function as those given above for the command **current_edges.**

References

1. K.S. Kunz and R.J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics,* CRC Press, 1993.

2. H. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Engineering,* Vol. 2, No. 2/3, pp. 135-148, 1991.

Technical Information Department · Lawrence Livermore National Laboratory
University of California · Livermore, California 94551