

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0376524 5

ORNL/TM-12359

oml

**OAK RIDGE
NATIONAL
LABORATORY**

MARTIN MARIETTA

**The TORSED Method for Construction
of TORT Boundary Sources from
External DORT Flux Files**

W. A. Rhoades

OAK RIDGE NATIONAL LABORATORY
CENTRAL RESEARCH LIBRARY
CIRCULATION SECTION
4500N ROOM 175

LIBRARY LOAN COPY

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this
report, send in name with report and
the library will arrange a loan.

©ORNL/MS/877

MANAGED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; prices available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5205 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-12359

406

37

38

Engineering Physics and Mathematics Division

THE TORSSED METHOD FOR CONSTRUCTION
OF TORT BOUNDARY SOURCES FROM
EXTERNAL DORT FLUX FILES

W. A. Rhoades

DATE PUBLISHED - AUGUST, 1993

Prepared by
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U. S. Department of Energy Systems, Inc.
under Contract No DE-AC05-84OR21400



3 4456 0376524 5

TABLE OF CONTENTS

	Page
LIST OF FIGURES	v
ABSTRACT	vii
SECTION I (THE TORSSED CODE)	1
1. INTRODUCTION	2
2. THEORETICAL INFORMATION	3
2.1 Basic TORSSED Method	3
2.2 Accuracy Considerations	3
2.3 Alternate Formulations -- The "Look-Forward" Method	4
2.4 Alternate Formulations -- The DOTTOR Method	4
3. PROGRAMMER'S INFORMATION	9
3.1 Input Output, and Common Blocks	9
3.2 General Code Structure	10
3.3 Supporting Service Routines	11
4. INSTALLATION AND ENVIRONMENT INFORMATION	14
4.1 Installation	14
4.2 Data Files	14
4.3 System Requirements	15
5. USER INFORMATION	16
5.1 Input Requirements	16
5.2 Printed Output	16
5.3 Error Messages	17
6. DEMONSTRATION PROBLEMS	18
APPENDIX A - TORSSED INPUT REQUIREMENTS	20
APPENDIX B - COMMON BLOCK STRUCTURE	22
APPENDIX C - PRINCIPAL DATA ARRAYS	23
APPENDIX D - DISCUSSION OF THE 3-D DISCONTINUOUS MESH TECHNIQUE ...	25
APPENDIX E - TORT BOUNDARY FLUX FILE FORMAT	31

SECTION II (THE VISA CODE)	35
1. INTRODUCTION	36
2. THEORETICAL INFORMATION	37
3. PROGRAMMER'S INFORMATION	38
3.1 Input, Output, and Common Blocks	38
3.2 General Code Structure	38
3.3 Supporting Service Routines	39
4. INSTALLATION AND ENVIRONMENT INFORMATION	40
4.1 Installation	40
4.2 Data Files	40
4.3 System Requirements	40
5. USER INFORMATION	41
5.1 Input Requirements	41
5.2 VISA Without GRTUNCLE Data or With GRTUNCL Data Only	42
5.3 Special Group Options	42
5.4 Synthetic Flux	42
5.5 Printed Output	42
5.6 Error Messages	43
APPENDIX A - VISA INPUT REQUIREMENTS	44
APPENDIX B - COMMON BLOCK COMVIS	45
APPENDIX C - VISA OUTPUT FILE FORMAT	47
APPENDIX D - DORT FLUX MOMENT FILE FORMAT	50
APPENDIX E - DORT SOURCE MOMENT FILE FORMAT	53
APPENDIX F - "DOT ANGULAR FLUX TAPE" FORMAT AS USED BY DORT	56
SECTION III (REFERENCES)	60
REFERENCES	61

LIST OF FIGURES

Figure		Page
2.1	Transformation From an RZ DORT to an XYZ TORT	6
2.2	Spatial Interpolation to a Point on the TORT Surface	7
2.3	DOTTOR Method of Direction Mapping	8
3.1	Radial Location of TORT Surface Point in DORT Cylindrical Coordinates	13
D-1	The Discontinuous Mesh Feature Allows the Mesh to be Locally Dense Where Required to Detail Problem Features	26

ABSTRACT

The TORSED method provides a means of coupling cylindrical two-dimensional DORT fluxes or fluences to a three-dimensional TORT calculation in Cartesian geometry through construction of external boundary sources for TORT. This can be important for several reasons. The two-dimensional environment may be too large for TORT simulation. The two-dimensional environment may be truly cylindrical in nature, and thus, better treated in that geometry. It may be desired to use a single environment calculation to study numerous local perturbations.

In Section I, the TORSED code is described in detail, and the diverse demonstration problems that accompany the code distribution are discussed. In Section II, an updated discussion of the VISA code is given. VISA is required to preprocess the DORT files for use in TORSED. In Section III, the references are listed.

SECTION I

THE TORSED CODE

1. INTRODUCTION

The purpose of TORSED is to provide a quick, reliable coupling between 2-Dimensional (2-D) DORT [rh88] calculations in cylindrical (RZ) geometry and 3-Dimensional (3-D) TORT [rh91b,rh91a,rh87] calculations in Cartesian (XYZ) geometry. There can be several motivations for such a coupling. In some cases, the entire environment is too large for 3-D simulation. An example is a building located on the ground a kilometer from a weapon source. In other cases, a portion of the problem is cylindrical in nature, but 3-D geometry is required for local detail. An example is a fixture at the surface of a reactor pressure vessel. In still other cases, fine mesh spacing or a special directional quadrature is required in the 3-D case that is not required in the 2-D environmental calculation. It may also be desired to study numerous local configurations with a single environment calculation.

An earlier code, DOTTOR, [th86] was constructed more or less concurrently with TORT to perform the mapping of the air/ground environment fluence from a weapon source to the surface of a concrete building. This was used successfully in a detailed study of radiation received inside the building. [rh92b,rh89] DOTTOR was also used in a study of detailed flux patterns near beam tubes in the High Flux Isotope Reactor (HFIR). [ch88b,ch88a]

The details of DOTTOR's construction made it relatively expensive to use, however, and extending it to problems using one or two million mesh cells and discontinuous-mesh geometry was not feasible. In addition, questions about the reliability of DOTTOR have arisen over the years. By comparing DOTTOR results with results from the new TORSED code, certain malfunction modes have been identified, and error stops have been put in place to prevent them. With these changes, DOTTOR can apparently be used reliably, within its scope of applicability, and it was valuable in checking the early TORSED results.

TORSED uses a much simpler procedure to map fluxes or fluences from one direction set and geometry to another, and it is constructed to run very large problems in minimal time and computer memory. It is fully compatible with the discontinuous-mesh features of TORT. Its straightforward construction facilitates checking and maintenance. Its simple procedure has proven adequate for cases where the same quadrature is used in 2-D and 3-D calculations. A few special cases could, theoretically, prove inaccurate, and those will be discussed later.

At this time, TORSED is compatible with all TORT features except the option to vary directional quadrature by energy group. If that should prove valuable, it can be added at a later date. Alternate procedures for performing the directional remapping are discussed in a later section. The early applications have been quite successful, however, and the demand for considering alternatives is not urgent.

2. THEORETICAL INFORMATION

2.1 Basic TORSED Method

The task accomplished by TORSED is quite straightforward. It is to read the fluence files from an RZ DORT calculation and to prepare an external boundary source file for a TORT XYZ calculation (Figure 2.1). As indicated in the figure, the DORT direction set rotates with the azimuthal variable, rather than remaining fixed in space. The vertical plane in the illustration indicates the RZ grid of the DORT problem. In 2-D cylindrical geometry, there is no azimuthal grid, of course, and the same fluence applies at every azimuth. The TORT geometry is located with respect to the DORT geometry by the radius and height of its origin, and by the rotation of its X axis counterclockwise from the DORT R axis. The Z axes of the two problems are assumed to remain parallel. The TORT direction set is fixed in space throughout the entire TORT geometry, but, in general, none of the directions match DORT directions exactly.

The spatial interpolation is straightforward. The average DORT flux for each volume cell is assumed to be the flux value for the geometric center of the cell, and the flux at the geometric center of each surface cell on the TORT geometry is assumed to apply to that entire TORT cell. Accordingly, the radius and height in DORT coordinates of each center on the TORT surface is obtained, and the flux at that point for each DORT direction is obtained by linear spatial interpolation between the nearest-neighbor DORT cell-center flux values (Figure 2.2). Linear interpolation of the logarithm of DORT flux is also an available option. It may be noted that either interpolation requires that the DORT mesh be large enough that its mesh centers completely enclose the TORT geometry. Extrapolation can lead to serious errors, and it is not allowed.

The spatial interpolation establishes a value for the flux in each of the DORT directions at each cell center on the TORT surface. Some type of directional remapping must be used to obtain the flux in each TORT direction. TORSED presently uses a "look-backward" method in which the flux in each TORT direction is set equal to the DORT flux in the nearest DORT direction.

2.2 Accuracy Considerations

The spatial interpolation requires a grid sufficiently fine near the TORT surface that linear interpolation of the flux is valid. When a cylindrical surface is modeled, it must be remembered that the Cartesian representation of this surface will be jagged.

The look-backward method used in directional interpolation is at its best when matching direction sets are used in DORT and TORT, and when the DORT flux varies smoothly over the direction space. Examples of cases not properly treated by this method are the collapse of a fine DORT quadrature into a coarse TORT quadrature and the case where a single DORT ray such as the ray containing the uncollided component in an air transport environment has a large value. In the case of a single large value, it is also important that the DORT directions have equal weight, since no correction for weight mismatch is available.

A less obvious limitation can arise when the contents of the enclosed TORT volume do not have the same reflection and transmission properties as the corresponding space in the original DORT problem. If the flux entering the surface is perturbed significantly by changes within the volume, then the result may not be correct. This can happen, for example, if a large volume of air in the DORT geometry is replaced with a scattering medium that would reflect particles. If those reflected particles are able to leave the enclosed volume and then re-enter it, the incoming DORT flux would not be correct for that case. Similarly, if particles would normally have passed through the volume and would have scattered back into it, new material in the volume might prevent that. Removing material can also cause perturbation. For example, replacing water with a beam tube removes reflection that would have taken place, and both outgoing and incoming flux may be altered.

Several situations can mitigate this problem, however. If the enclosed volume is small with respect to a mean-free-path in the surrounding medium, a particle is unlikely to enter, leave, and then re-enter the volume. Backscatter of transmitted flux is rarely important due to the low probabilities involved. In the beam tube example, an unperturbed surface can be selected several mean-free-paths away from the perturbation in the water. If a small perturbation is described, selecting a surface distant with respect to the largest dimension of the perturbation may result in an unperturbed surface. It may be noted that perturbation of the outgoing flux is not important, since TORSSED deals only with the incoming flux.

2.3 Alternate Formulations – The "Look-Forward" Method

One alternative to the "look-backward" method would be a "look-forward" approach. In this method, each incoming DORT ray would be apportioned among several nearest-neighbor TORT rays according to the relative nearness and weight of the neighbors. This method would account for a large discrete ray correctly, and it would allow a fine quadrature to be mapped into a coarse. It would be fairly tolerant of mismatched direction sets and uneven direction weights. It would always preserve flux and, if the direction sets were sufficiently fine, it would approximately preserve current. In certain special cases, however, it might provide no flux at all in some of the TORT directions. It would fail badly if a coarse DORT direction set were mapped into a fine TORT set. It would probably require the calculation and storage of additional information to be used in the mapping. Its use in other codes has not been so common as use of the "look-backward" method.

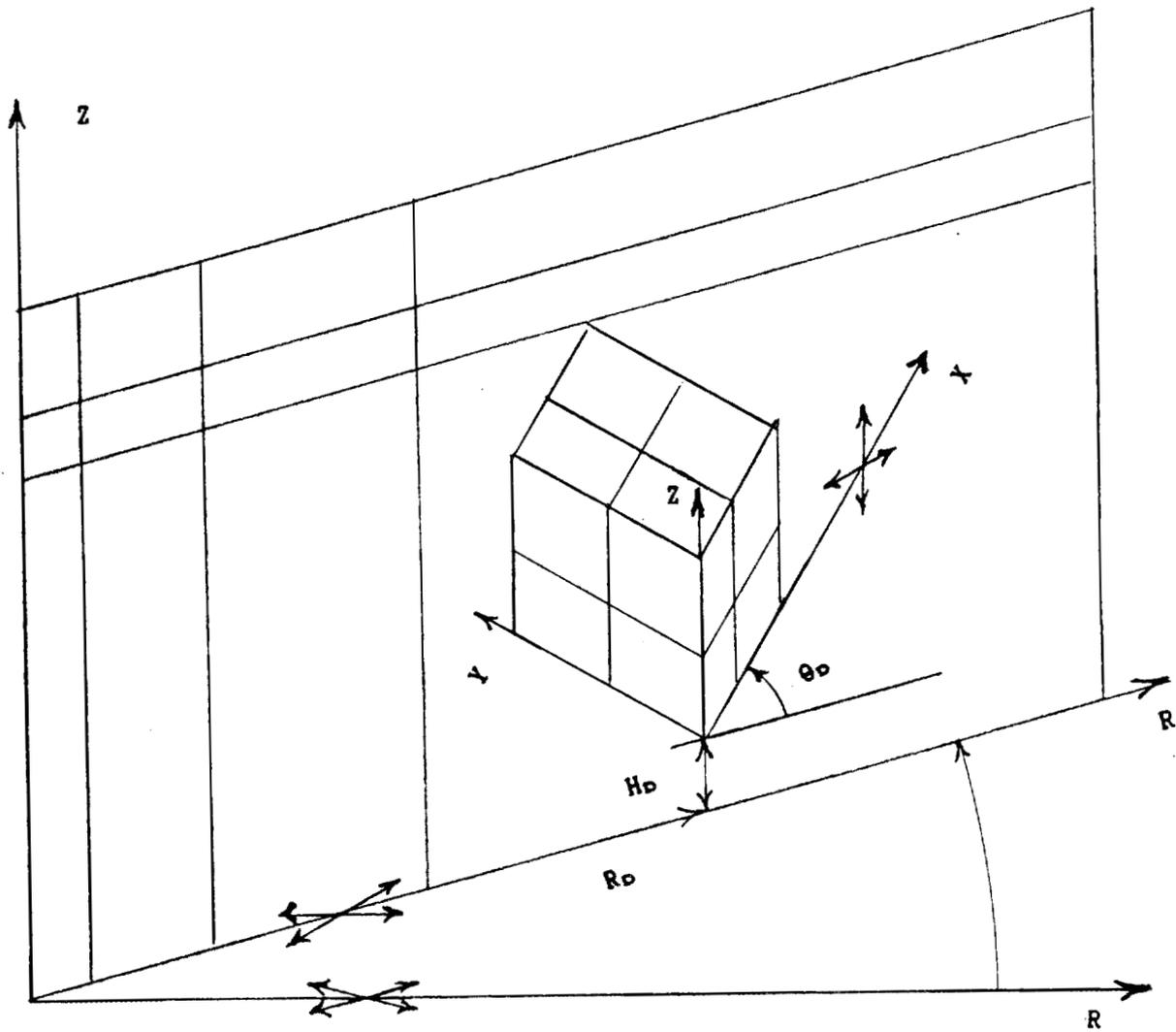
2.4 Alternate Formulations – The DOTTOR Method

We will describe briefly, for the purpose of comparison, the method used by DOTTOR. Full details can be found in the reference. DOTTOR establishes sectors of direction space corresponding to each DORT and each TORT direction (Figure 2.3). Although the basic discrete ordinates formulation does not guarantee that this is always valid, it is a workable plan with the direction sets in common use with DORT and TORT. The DORT flux is assumed to be constant over its sector. Since the coordinates of the sectors are the azimuthal angle and the cosine of the polar angle, then summing each DORT flux times the area of its intersection with a given TORT sector approximates the integral of

DORT flux over the sector. Dividing by the TORT sector area gives the remapped flux to be used as a source in the TORT problem.

This method generally has all of the desirable traits of the look-forward and look-backward methods except simplicity. It proved to be quite tedious to program and debug. The number of remapping constants is potentially huge; potentially, there would be a different set of coupling coefficients at each surface cell of the TORT geometry. Accordingly, the constants were calculated each time they were used, causing very large problems to require undesirable amounts of computer time. Still, the method performed well with all combinations of quadratures, treated discrete rays correctly, provided non-zero source in all directions, and preserved flux. It was a valuable basis for comparison in the present work.

Figure 2.1 -- Transformation From an RZ DORT to an XYZ TORT



R_D = radius of TORT origin in DORT geometry

H_D = height of TORT origin in DORT geometry

θ_D = rotation of TORT origin in DORT geometry

Figure 2.2 -- Spatial Interpolation to a Point on the TORT Surface

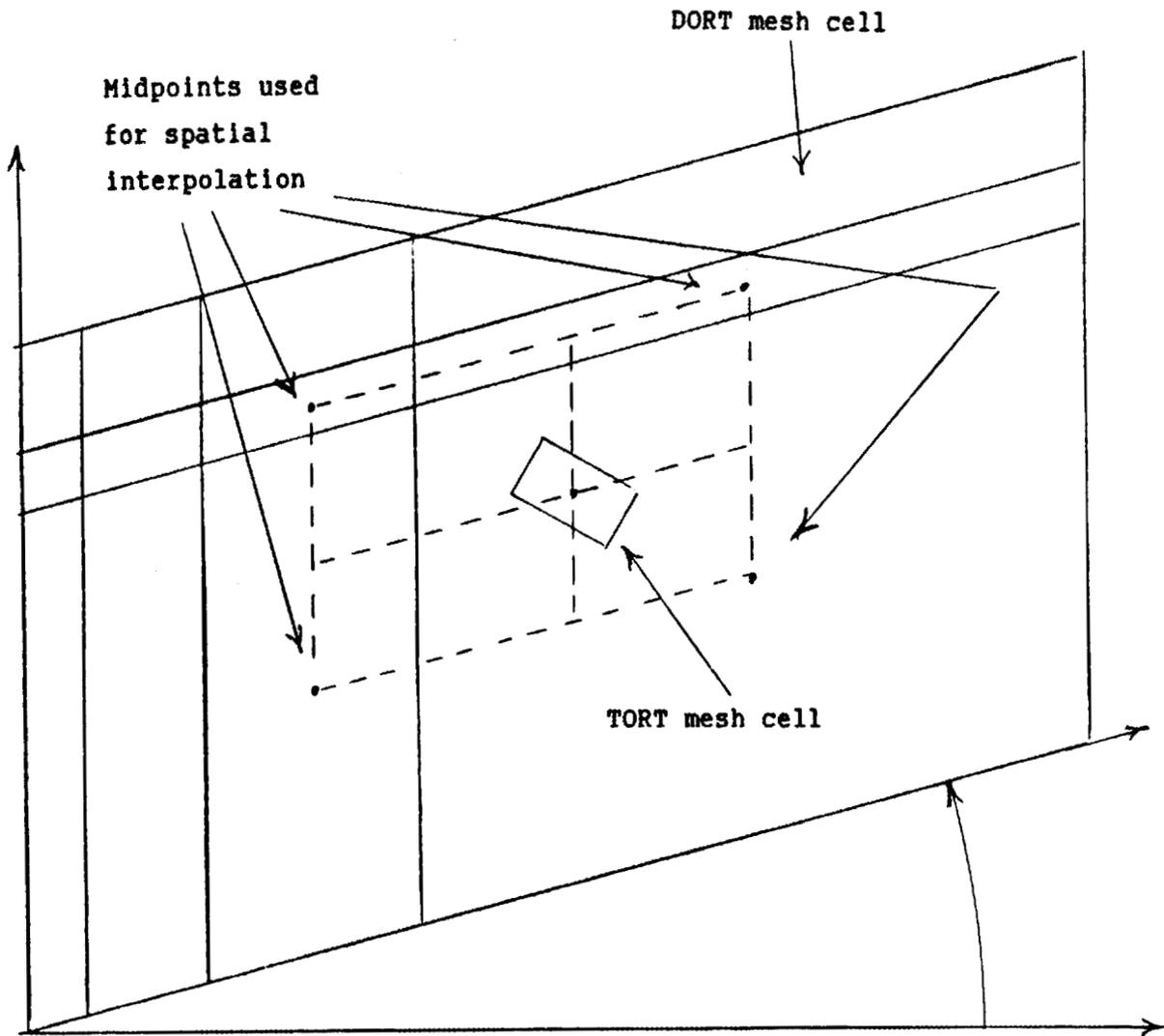
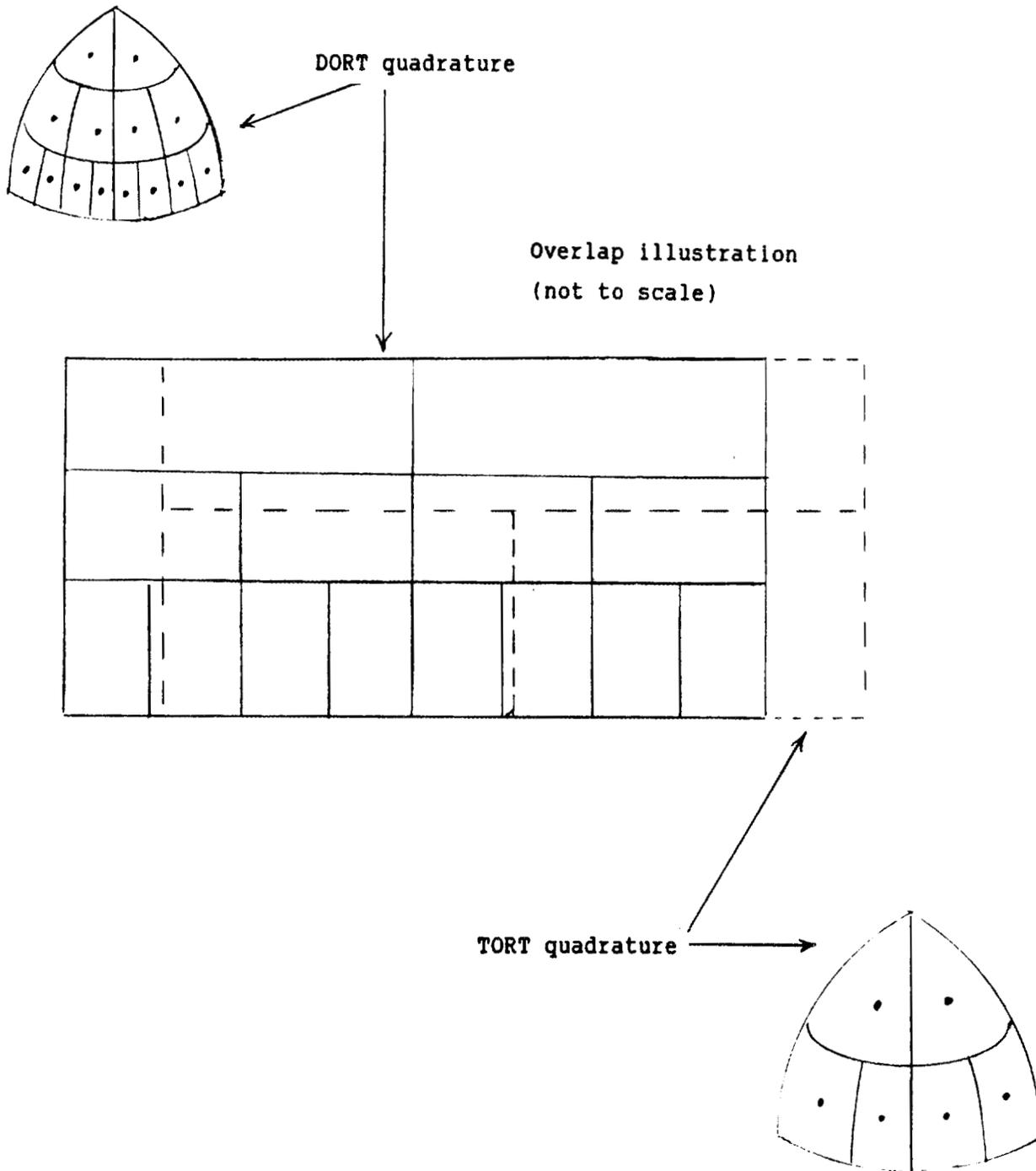


Figure 2.3 -- DOTTOR Method of Direction Mapping



3. PROGRAMMER'S INFORMATION

3.1 Input Output, and Common Blocks

The task of TORSED, from a programmer's point of view, is to transform an input file containing DORT flux values as processed by the VISA [rh74a] code into an output file for use as a boundary source by TORT. The format of the input file is that produced by the VISA code with the NTYPE parameter set to 1. A description of the VISA code should be consulted for detailed specifications. The output file is produced in the VARBND format used as input by the TORT code. The VARBND format is described in Appendix E.

Certain parameters and data arrays are required as input, as specified in Appendix A. In addition to the job title and data required to control the execution, descriptions of the space mesh and directional quadrature are required. The input parameters are read directly into common block COMSED described in Appendix B. The arrays are read into a single large "container" whose location is determined by calls to DORT support subroutines.

The relative location of each individual array in the container is defined by a pointer stored in the common block. The pointer for array ?? is always named L??, etc. For example, the array IVAL begins at a location in the container array specified by the pointer LIVAL. In general, the pointers are stored in common in ascending order, so that the difference between the values of successive pointers indicates each array length. The container, once located, is called D. Then, to use array IVAL in a subroutine, the argument D(LIVAL) is included in the call statement, and the name IVAL is used in that position in the subroutine statement. This method of data storage is sometimes called "flexible dimensioning". It is similar in concept to the pointer feature of Cray FORTRAN, but it is not machine dependent. Since compilers do not require a special treatment of the flexible dimensioning array, there is no loss of efficiency comparable to that experienced with the Cray pointers.

Additional input parameters are obtained from the input files as indicated in the first section of Appendix B. The appendix also indicates a number of parameters generated internally by the code as noted.

TORSED uses several common arrays from DORT in order to communicate with DORT support subroutines. These blocks are:

COMIN -- general job status information
COMIO -- I/O unit status information
COMBLK -- reference address for the container

The general ordering of data arrays in the container is indicated in Appendix C. Several arrays there are listed as originating from the user input or from the VISA input file. In certain special cases, data arrays may overlay each other, if they are not used concurrently, of course. The source listing is the final word on those details.

If $IMTL.gt.0$, the source arrays numbered 7? are not supplied, and only the "t" terminating that block is to be input. In that case, a conventional continuous mesh source is constructed for TORT. If $IMTL.lt.0$, a discontinuous mesh source will be constructed, and the 7? arrays are required as described in Appendix A. A detailed discussion of the discontinuous mesh technique is given in Appendix D. Briefly the number of space cells is allowed to vary from row to row in a plane, and the number of rows is allowed to vary from plane to plane. This feature allows the computational work to be concentrated where it is needed to detail special features, and it is possible to define very complex geometric structures in this way. An important advantage of the feature is that it does not require the abandonment of the mathematical solution procedures developed for continuous mesh problems.

3.2 General Code Structure

A walkthrough of the main structural subroutines serves to describe the code structure.

MAIN -- the main program reads the input data, obtains computer memory for the container, and calls **SEDIN**.

SEDIN -- **SEDIN** calls **DOPC** to initialize the I/O process and then sets up the pointers required for data array storage. It calls **INPA** twice, reads the remainder of the arrays, and continues the problem solution by a call to **SEDUM**. At the conclusion, a final call to **DOPC** disposes of all open data files properly.

INPA -- the first call to **INPA** counts the total number of rows in the TORT mesh. In a continuous-mesh problem, this is simply $JM*KM$. The second call to **INPA** fills arrays needed to support the discontinuous mesh process, and then identifies K-sets. A K-set is a set of planes having identical space mesh specifications. The first plane belongs to the first K-set by default, as do all planes like it. The next plane to have a different space mesh begins K-set 2, and so on. The K-mate of a plane is K for the first plane having the same space mesh, i.e. the first plane having the same K-set value as the plane in question.

SEDUM -- **SEDUM** calls the TORT quadrature and geometry routines, reads the description of the DORT quadrature and geometry from the VISA input file, calls the DORT quadrature and geometry routines, and then calls **FLUXRZ** to complete the execution.

QUADT -- the TORT quadrature routine supplies the missing cosine, **XZIT**, and performs consistency checks to assure that the quadrature set is valid.

GEOMT -- the TORT geometry routine finds the width and midpoint of each mesh interval and performs consistency checks on the mesh.

QUADD -- the DORT quadrature routine supplies the missing cosine, **XZI**, calculates the cylindrical coordinates, **RAD** and **PHI**, of the projection of each direction in the plane perpendicular to Z, and calculates a "double-DORT" quadrature set in which the outward and inward directions are separated. It then uses subroutine **ROTATE** to find the double-DORT

direction matching each TORT direction at the position of the TORT origin and to convert the double-DORT direction index to a standard DORT direction index. This index, MATCH, is listed in the standard edit of input data.

GEOMRZ -- the DORT geometry routine obtains the cylindrical DORT coordinates of each cell center on the surface of the TORT geometry. The arrays are called RZ? and THZ?, where ? takes on the values L, R, I, O, B, and T denoting the TORT left, right, inside, outside, bottom, and top faces. Considerable reliance is placed on the built-in trigonometric functions in order to simplify the work. Figure 3.1 illustrates the logic used in the transformation. The special function ATANX is simply a call to the built-in function ATAN2 with special provision made for the case where both arguments are 0. Some machines handle this case gracefully, providing 0 as the value, and others do not. ATAN2 always provides the azimuth in the proper quadrant for use with other functions, which is a considerable convenience. A degenerate case occurs when the position being converted is much farther away from the TORT axis than the TORT axis is distant from the DORT axis. In that case, the DORT cylindrical coordinates are taken to be equal to the TORT coordinates with little error. This procedure was tested extensively for various inputs, and it proved quite robust in spite of the double application of the law of cosines.

3.3 Supporting Service Routines

TORSED draws extensively from the collection of service routines from the DORT distribution. A brief discussion of the routines follows. In general, the source listing of each routine provides more detailed information as to the use and proper calling sequence. In some cases, the routines call other routines not discussed here, but those have the same function as in DORT.

BLKIO -- controls the reading and writing of random (direct) access scratch arrays.

CLEARX -- sets a string of data locations to 0.

CSETI -- sets a string of data locations to a value supplied by the user.

DLOCAL -- after initialization, acquires a memory area from the system for the container array, and later, returns it.

DOPC -- controls the opening, and closing of random (direct) access scratch files.

ERRO -- writes error message, records highest error code reached, and provides termination if error code is too large.

FIDOS -- implements the FIDO data input format shared with DORT and TORT.

HEADER -- writes a heading for the job.

NDXR -- assembles the pivot arrays for discontinuous mesh problems.

SEQIO -- manages the opening, reading, writing, and closing of sequential disk files.

TIMEX -- provides summary of incremental and cumulative computer usage charge.

TIMSET -- initializes the timing function performed by TIMEX.

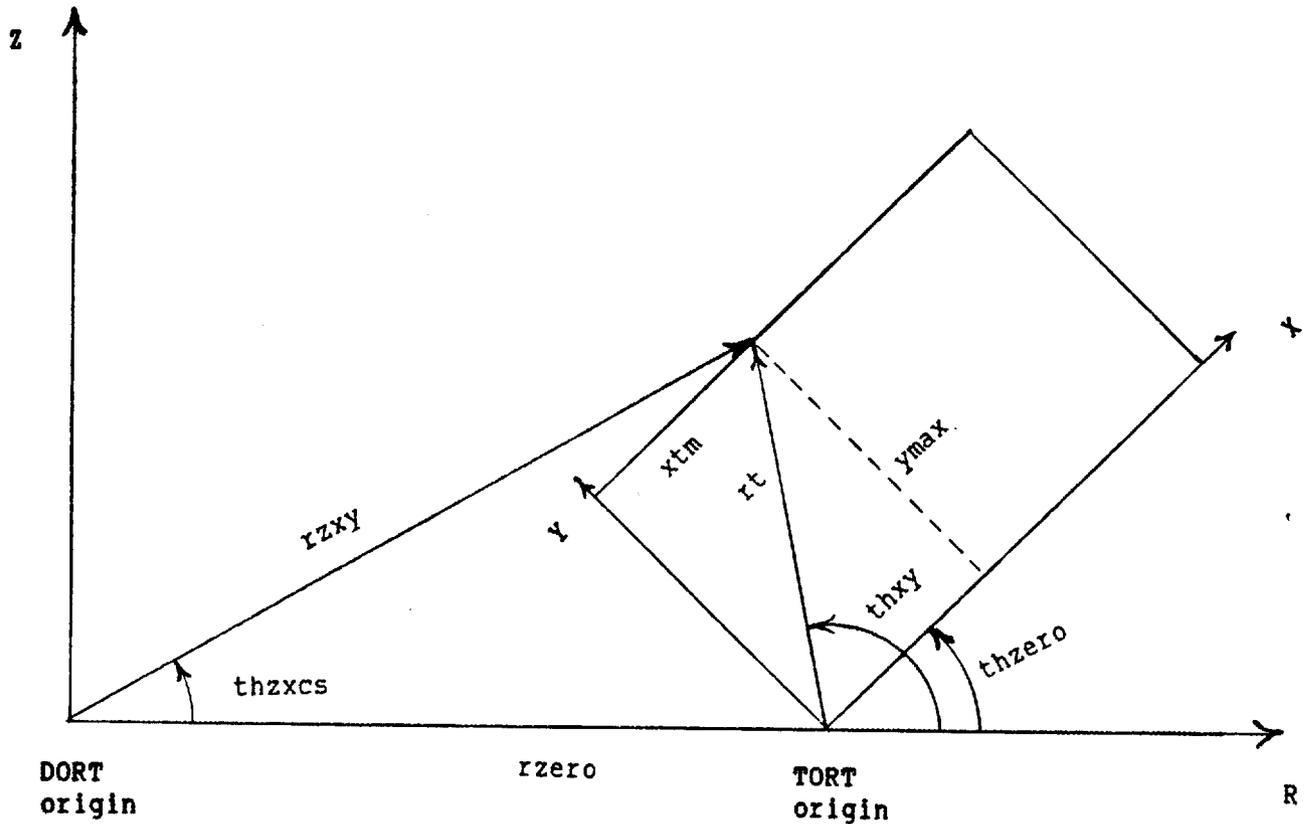
WANDR1 -- performs special I/O tasks in reading the VISA input file.

WOT10 -- provides edit of parallel columns of data, where the columns may not be of the same length.

WOT4 -- provides edit of multidimensional arrays.

Figure 3.1 -- Radial Location of TORT Surface Point in DORT Cylindrical Coordinates

(Illustration depicts an xy surface; dort z location = tort z + zzero)



```

thxy = thzero + atan2( ymax, xtm)           # TORT azimuth
rzero2 = rzero*rzero
rt2 = xtm*xtm + ymax*ymax
rt = sqrt( rt2)                             # TORT radius
rzxy2 = rzero2 + rt2 + 2*rzero*rt*cos(thxy) # law of cosines
rzo = sqrt( rzxy2)                          # DORT radius
thzxc = (rzero2 + rzxy2 - rt2) / (2*rzero*rzo) # cosines again
thzo = sign( acos( thzxc), sin( thxy) )     # DORT azimuth

```

4. INSTALLATION AND ENVIRONMENT INFORMATION

4.1 Installation

The installation is assumed to be on a computer using a UNIX system. The reason for this is that UNIX is the only system available to the developer. Both DORT and TORT are installed using C-shell scripts contained as a part of the distribution material. The user designates the machine to be used, and the unloading procedure selects the appropriate installation scripts, in addition to making minor adjustments to the source material as required. The adjustments involve selecting alternate paths to use special features such as vector loops and selecting alternate subroutines to perform machine-specific system tasks such as timing.

The materials required for both VISA and TORSED are made available as DORT is unloaded. After installing DORT and before installing TORT, VISA and TORSED can be installed with the following command:

```
csH -x -S jcdor visa
```

This also creates a c-shell script "JSED" to be used later.

After installing TORT using the JCLTOR script, standard test problems can be run by executing:

```
csH -x -S jsed run xxx
```

where xxx is the name of a problem set chosen from:

```
dog2  
dogag  
ag  
areac
```

The output will appear on a file named oxxx.

4.2 Data Files

Standard UNIX naming conventions are followed, in that the file associated with logical unit ?? is named fort.?. The standard input is assumed to be unit 5, and the standard output is unit 6. Those files are assumed to be opened by the system without explicit action by the program. The two sequential data files containing the VISA flux input and the TORT source output may be any number between 1 and 80 except 5 and 6.

The code opens scratch files on 81, 82, 83, and 84. These are normally opened as random (direct) access files, but sequential files could probably be substituted with some loss of efficiency. The size of each file is:

name	number	words	use
NTFLXD	81	MM*IM*2	DORT flux for two adjacent rows
NTMIK	82	MMT*JMT*KSM	MIO array (if IMTL.lt.0)
NTMJK	83	MMT*IMT*KSM	MJO array (if IMTL.lt.0)
NTMIJ	84	MMT*IMT*JMT	MKO array

4.3 System Requirements

The codes are intended to be operable on any system on which DORT can be installed. The TORSED routines are all written in FORTRAN 77, and they are in frequent use on both Cray mainframes and IBM workstations. The DORT routines make use of certain system-dependent features to provide special capability such as run-time memory allocation and timing data. Some C language is used in that area. Generic routines are provided with DORT that should allow operation with somewhat reduced capability on systems for which no specific compatibility package exists.

The memory and CPU usage have been minimized by the use of scratch files. A very large test problem generated 20 groups of source for a 100,000 mesh cell problem on a Cray Y-MP while using only 36,584 words of memory and 0.2 minutes of CPU+SYSTEM time. The delay produced by the scratch file usage was not measurable.

5. USER INFORMATION

5.1 Input Requirements

The input data requirements are detailed in Appendix A. With the exception of the alphanumeric title, all data are input using the FIDO format also used by DORT and TORT. The user may refer to the corresponding documents for a specification of that format.

The unit numbers of two files are required:

NVISA -- file produced by VISA on unit NDATA

NTORT -- boundary source prepared for use by TORT on unit NTBSI

The default memory allocation is sufficient for all but the largest problems. If more is required, it can be requested by supplying a value for LOCOBJ.

The default value of NEDIT is adequate for most uses. If a programmer requires more output for diagnostic purposes, one of the sample problem illustrates that procedure, and the resulting output is discussed later. This is never required in normal use.

5.2 Printed Output

The TORSED title and integer input parameters are edited first, followed by titles and parameters from the VISA file. Next, the TORT quadrature, and space mesh are edited, together with J-set, K-set, and K-mate data needed when the discontinuous mesh option is used. This is followed by the DORT quadrature and space mesh data. It may be noted that the R and Z midpoints listed are only those for which fluxes have been transferred to TORSED. An array of integers relates the R midpoints to the corresponding DORT intervals. The meaning of PHI and RAD is explained in the earlier discussion of subroutine QUADD.

This is followed by a listing of the full double-DORT quadrature, followed by the azimuthal angle and cosines of the DORT directions with respect to the TORT coordinate set at the location of the TORT origin. A column of integers gives the DORT direction corresponding to each TORT direction at that location.

After each group is processed, a message containing the group index and the last upward flux value for the last four intervals in the last row of that group are given as a rough indication of the results.

If NEDIT.gt.0, a large amount of diagnostic print is given, largely useful for debugging the program. Briefly, this consists of the DORT coordinates of each TORT surface cell and the DORT direction matching each TORT direction at each location. If N*10 is added to nedit, the value of each flux that becomes the source for TORT for groups 1 through N will be printed. Further details can be found by inspecting the program source.

5.3 Error Messages

Certain conditions can produce error warnings and, if severe enough, a halt to execution. Those from SEDIN, QUADT, GEOMT, GEOMRZ, and INTERP are self-explanatory. Certain other error messages can arise from the DORT service routines, and these have the same meaning as in DORT.

6. DEMONSTRATION PROBLEMS

Several demonstration problems have been developed to illustrate the use of VISA and TORSED to link DORT to TORT. The input streams for the problems are available in the TORT distribution material. Each of the problems uses the JDOS driver procedure as used with certain of the DORT and TORT problem sets. Instructions can be found in the distribution material for DORT. A discussion of each of the VISA-TORSED demonstration problems follows.

ODOG2 --

A "metric doghouse" 2m x 1m x 1m high, with 5cm walls and 10cm roof of an absorbing material, is situated on the ground at a ground range of 205cm from a point source. This range was chosen so that the doghouse subtends roughly a 90-degree azimuthal angle in the horizontal plane, a severe test of the directional remapping. The source is located at a height of 300m. Fictitious 2-group cross sections are used. The TORT solution is done with P1 scattering and S2 symmetrical quadrature.

A continuous-mesh 2-group source for the doghouse is obtained from TORSED, and then, a discontinuous-mesh source is obtained. Two TORT problems demonstrate the use of the continuous- and discontinuous-mesh sources in turn. The results have been studied in considerable detail. The key responses shown indicate some differences in pointwise results, partly due to the fact that the midpoint locations have shifted, but the region integrals agree quite closely. These and other comparisons indicate correct functioning of the discontinuous mesh feature.

ODOGAG --

The cross sections used in TORT problem set 6 are used to give a 20-group air-ground environment for the metric doghouse. The doghouse was moved to a ground range of 468m and rotated 270 degrees so that radiation could stream directly in through the doorway. An S6P3 treatment was used for both DORT and TORT. The source was processed in 2 groups, but only the first group was solved with TORT. It may be noted that using the "alternate s8 quadrature from JVP" in the DORT portion of this problem will result in relatively poor results due to a mismatch in the weight and direction cosines applicable to the ray containing the uncollided flux. This difficulty could also occur in a shield with streaming ducts. In that case, a single ray or a few adjacent rays may carry a large fraction of the total flux. In such a case, it is important that the quadratures used in DORT and TORT match, and that the weights be uniform.

OAG --

All groups of the 20-group air/ground problem are solved on the Cray by DORT and TORSED. The TORT problem models an actual concrete building roughly 17m by 70m by 15m high. More than 100,000 mesh cells are required. On a Cray Y-MP, TORSED uses 0.18 minutes for 20 groups. The memory requirement is less than 37,000 words.

OAREAC --

A reactor problem demonstrating the detailing in XYZ geometry of a sector of a pressure vessel is solved. After DORT uses an S6P3 solution to establish the fluxes in RZ geometry, TORSED and TORT calculate the fluxes in an XYZ sector of the geometry just outside of the core. The agreement with DORT at the inner boundaries is within 7%, while 12% agreement is obtained at the outer boundary, where the curved surface of the core and container are represented by a jagged surface in XYZ. Mesh refinement near the coupling surface has been shown to produce an even closer match.

APPENDIX A -- TORSSED INPUT REQUIREMENTS

A.1 Title

A single line of identifying information (72 characters).

A.2 Parameter Input Block

61\$\$ -- Integer Parameters

nvisa = visa input unit number
ntort = tort output unit number
imti = no. tort i intervals; (neg=discont. mesh)
jmt = no. tort j intervals
kmt = no. tort k intervals

mmt = no. tort directions
nedit = edit control (use 0)
locobj= memory objective, words*1000
isp1 = spare; enter 0
isp2 = spare; enter 0

[finish this array with "e"]

62** -- Real Parameters

rzero = dort radius of tort coordinate origin
zzero = dort height of tort coordinate origin
thzero= ccw rotation of tort coordinates (degrees)
flxmin= minimum flux for log interpolation (0: use linear interpolation)

[finish this array with 'e']

[follow these arrays with 't']

A.3 Discontinuous Mesh Block

71\$\$ iset [jmt*kmt entries] i set by row and plane
72\$\$ imbis [jmt*kmt entries] # of cells by i set
73\$\$ jset [kmt entries] j set by plane
74\$\$ jmbjs [kmt entries] # of rows by j set
75\$\$ mset [igm entries] m set by energy group
76\$\$ mmbms [igm entries] # of directions by m set

[follow these arrays with 't']

Notes:

- . Fill unused portions of arrays with 0.
- . Arrays 71-74 are to be entered if and only if imti.lt.0.
- . Arrays 75-76 have not been implemented yet.

A.4 Directional Quadrature Block

81** wt [mmt entries] weight by direction
82** emut [mmt entries] cosine of angle with x axis
83** etat [mmt entries] cosine of angle with z axis

[follow these arrays with 't']

A.5 General Data Array Block

```
1** xt      [entries: sum of imbis+1 over all i sets]  x mesh boundaries
2** yt      [entries: sum of jmbjs+1 over all j sets]  y mesh boundaries
3** zt      [entries: kmt+1]                          z mesh boundaries
```

[follow these arrays with 't']

Notes:

- . iabs(imti) must be the length of the longest i set.
- . jmt must be the length of the longest j set.

APPENDIX B -- COMMON BLOCK STRUCTURE

B.1 Parameters From VISA File

mm = number of directions in DORT quadrature
igm = number of energy groups
nip = number of radial points on VISA file
njp = number of axial points on VISA files

B.2 TORSED Input Parameters

[integer, then real parameters as listed in Appendix A]

B.3 Array Pointers

[pointers for each array in the data container; principal arrays are explained in Appendix C]

B.4 TORSED Control Parameters

ITEM	SET BY	DESCRIPTION
mm2	main	2*mm, number of double-DORT directions
mmdn	quadd	number of downward DORT directions
im	main	=nip, number of DORT i points
jm	"	=njp, number of DORT j points
mmdnt	quadt	number of TORT downward directions for an m-set
ksm	inpa	number of k-sets
imt	main	iabs(imti), max number of tort i intervals
imsism	sedin	sum of im over i-sets
jmsjms	"	sum of jm over j-sets
mmsmsm	"	sum of mm over m-sets
ism	"	number of i-sets
jsm	"	number of j-sets
msm	"	number of m-sets
imsjm	"	max number of ij cells in any plane
mmdut	quadt	max number of directions in any hemisphere, any m-set
ntflxd	sedin	scratch file for DORT flux
ntmjk	"	scratch file for mio array
ntmik	"	scratch file for mjo array
ntmij	"	scratch file for mko array
ifmij	main	=0: mjk, mik, mij stored on disk; =1: internally
jphold	interp	j of DORT plane previously read in
title	input	title of this torsed job or input VISA job
tdot	visa file	title of original DORT job

APPENDIX C -- PRINCIPAL DATA ARRAYS

ITEM	SET BY	DESCRIPTION
ival	VISA file	DORT i index of radial points
w	"	DORT direction weight
emu	"	DORT cosine with r axis
eta	"	DORT cosine with z axis
xzi	quadd	DORT cosine with theta axis
phi	"	DORT cylindrical geometry azimuth of direction m
rad	"	DORT cylindrical geometry radius of direction m
fw	"	double-DORT quadrature weight
femu	"	" cosine with r axis
fxzi	"	" cosine with theta axis
feta	"	" cosine with z axis
fphi	"	" cylindrical azimuth of direction m
frad	"	" cylindrical radius of direction m
rphi	"	" cylindrical azimuth of m in TORT system
remu	"	cosine of double-DORT direction with TORT x axis
rxzi	"	cosine of double-DORT direction with TORT y axis
match	"	index of DORT direction matching TORT direction mt
rm	VISA file	DORT radial position
zm	"	DORT axial position
iset	input	i set by j and k, padded with 0
imbis	"	im by i set
jset	"	j set by k, padded with 0
jmbjs	"	jm by js
mset	"	m set by ig
mmbms	"	mmt by m set
kset	"	k set by k
kmate	"	first k in k set ks
ibis	inpa	sum of imbis over is
jbjs	"	sum of jmbjs over js
ibjk	"	sum of ims(j'k) over j and k
ijbk	"	sum of cells per plane over k
jbk	"	sum of jms(k) over k
wt	input	TORT direction weight
emut	"	cosine of TORT direction with x axis
etat	"	cosine of TORT direction with y axis
xzit	quadt	cosine of TORT direction with z axis
xt	"	TORT interval boundaries on x axis
yt	"	" interval boundaries on y axis
zt	"	" interval boundaries on z axis
xm	geomt	" midpoint on x axis
ym	"	" midpoint on y axis
zm	"	" midpoint on z axis
xtd	"	" interval width on x axis
ytd	"	" interval width on y axis
ztd	"	" interval width on z axis

rzl	geomrz	radius of TORT point in DORT system, left surface
thzl	"	azimuth of TORT point in DORT system, left surface
r zr	"	radius of TORT point in DORT system, right surface
thzr	"	azimuth of TORT point in DORT system, right surface
rzi	"	radius of TORT point in DORT system, inside surface
thzi	"	azimuth of TORT point in DORT system, inside surface
rzo	"	radius of TORT point in DORT system, outside surface
thzo	"	azimuth of TORT point in DORT system, outside surface
rzb	"	radius of TORT point in DORT system, bottom surface
thzb	"	azimuth of TORT point in DORT system, bottom surface
rzt	"	radius of TORT point in DORT system, top surface
thzt	"	azimuth of TORT point in DORT system, top surface
ener		energy group boundaries
fluxd	VISA file	dort directional flux input
fio	fluxrz	tort directional source output
mio	geomth	m of DORT direction matching TORT mt, left/right surface
mjo	"	m of DORT direction matching TORT mt, in/outside surface
mko	"	m of DORT direction matching TORT mt, bottom/top surface

APPENDIX D -- DISCUSSION OF THE 3-D DISCONTINUOUS MESH TECHNIQUE

BASIC CONCEPT:

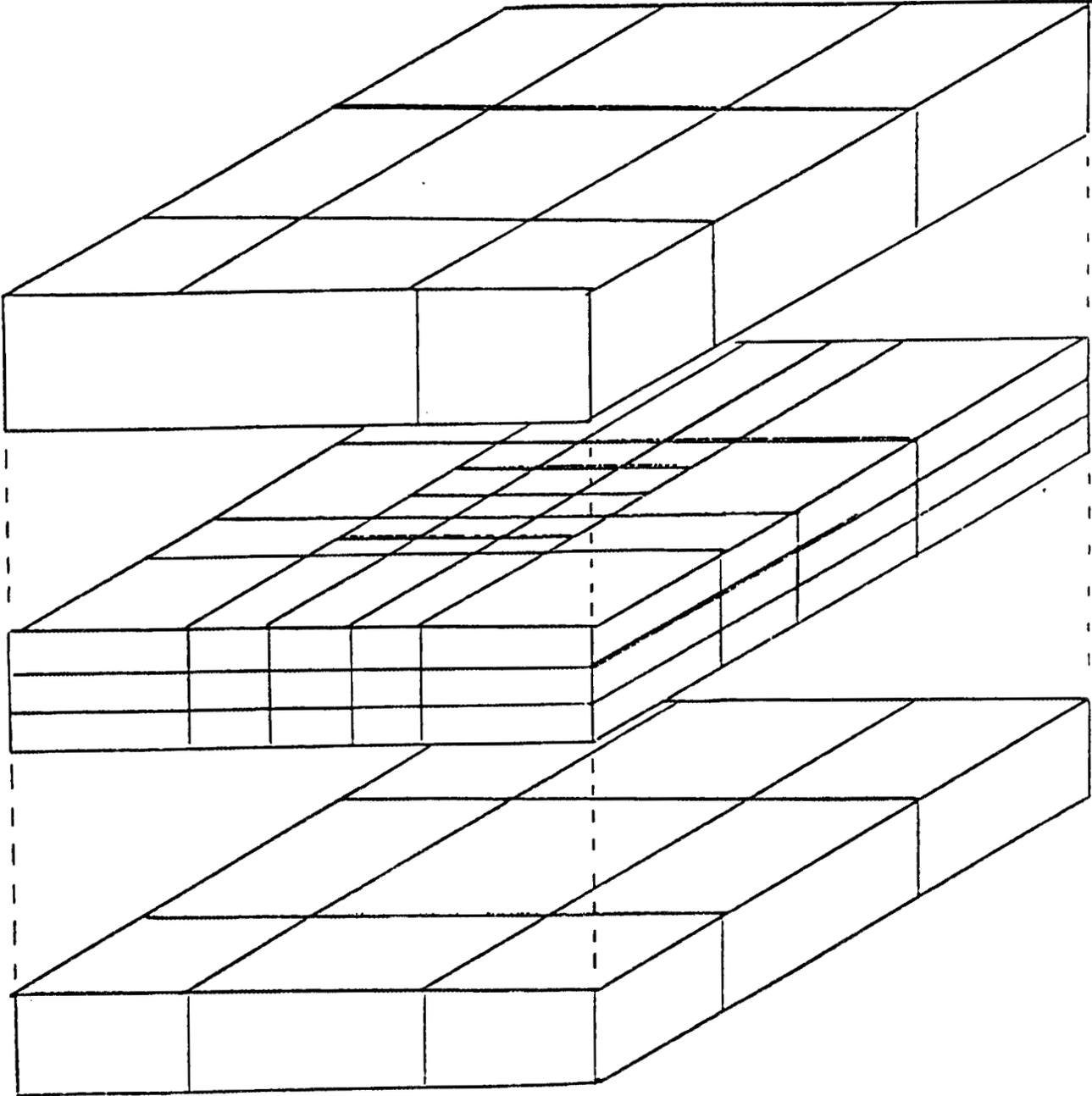
In a conventional continuous-mesh problem, a computational mesh is defined by mesh interval boundaries along each of the coordinate axes. Planes passing through these interval boundaries, perpendicular to the respective coordinate axes, define the surfaces of each cell. Opposite cell surfaces are always parallel, and they always meet adjacent surfaces at right angles. The cell surfaces run continuously through the mesh.

In such a mesh, let us call the first, second, and third coordinate axes the i , j , and k axes. The mesh cells lie in ordered rows parallel to the i axis, and the rows lie in planes perpendicular to the k axis. The vertical boundaries of each cell match the boundaries of adjacent cells.

In a discontinuous mesh, as the term is used here, the requirements are relaxed slightly. Mesh cells are still bounded by parallel planes, each perpendicular to one of the coordinate axes, and the planes meet at right angles. The new flexibility is that only the k boundaries, i.e. the boundary planes perpendicular to the k axis, are required to run continuously through the mesh. Thus, the other boundary planes may be discontinuous at intersections. The mesh cells lie in rows having common j boundaries, but their i boundaries need not match. Rows lie in planes sharing common k boundaries, but neither i nor j boundaries of adjacent planes need match (except as required to allow acceleration and at the outer boundaries of the problem space).

The advantage is that the mesh can be locally dense in areas where detail is needed most, thus using computational work more efficiently. An illustration is provided in Figure D.1. Since the transport within each cell is unperturbed by the irregularities, conventional evaluation procedures such as weighted difference, nodal, or characteristics methods can be used. Many years of research have gone into these methods, and they would not be relinquished easily.

Figure D.1 -- The Discontinuous Mesh Feature Allows the Mesh to be Locally Dense Where Required to Detail Problem Features



The conventional partial current acceleration already in widespread use in DORT and TORT is applicable in this instance, with the restriction that a coarse mesh be supplied, and that the coarse-mesh boundaries lie in each of the fine-mesh sets. (This last restriction could possibly be removed by reprogramming, but that has not been tried.)

The programming is significantly more complicated with the discontinuous mesh, but a system of pre-calculated "pivot arrays" allows data items to be located and used without measurable loss of efficiency. The pivot arrays will be discussed later. Since all of the mesh cells lie in rows, the computational sweeps performed by the conventional TORT/DORT subroutines can be used without modification, and they will run at the traditional speeds. Some computational work is required, of course, to perform the "remeshing", i.e. the remapping of boundary flows where adjacent rows and planes do not match.

An important advantage is that the system can be imbedded into a conventional code without disturbing the conventional operation significantly. Like DOT 4 and DORT before it, discontinuous mesh TORT can produce the expected results to a conventional problem, and at the expected cost.

Probably the most important disadvantage of the discontinuous mesh concept in two dimensions is that, although it can help in describing curved surfaces, it is not as powerful as general triangles or general quadrilaterals in this respect. Codes using the latter two concepts in discrete ordinates calculations exist, although none appear to have reached widespread use. It is not clear when they will be extended to three dimensions in a production code, or what the computational efficiency would be.

MACHINE IMPLEMENTATION:

We must necessarily ask the reader to pardon a mix of FORTRAN and algebra in that which follows. We will try to be clear.

First, we define:

IS \equiv index of an i-set, i.e. a set of x or r boundaries defining mesh cell surfaces; IS=1,...,ISM

JS \equiv index of a j-set, i.e. a set of y or theta boundaries defining mesh cell surfaces; JS=1,...,JSM

For j-sets, input arrays consist of:

JMBJS(JS) \equiv # of mesh cells in j-set JS
JSET(K) \equiv j-set number for plane K; K=1,...,KM

From these, we can always obtain:

JS = JSET(K), index of the j-set for plane K
JMS = JMBJS(JS), number of intervals for j-set JS

With regular indexing, where all JMBJS(JS) = JM, we (or the compiler) can locate a function of J and JS by a simple integer computation:

$$F(J,JS) = F(J+JM*(JS-1))$$

but, since JMBJS is not necessarily constant in a discontinuous mesh, we now define a "pivot array":

$$\begin{aligned} \text{JBJS}(1) &\equiv 0 \\ \text{JBJS}(JS+1) &\equiv \text{JBJS}(JS) + \text{JMBJS}(JS); \quad JS=1,\dots,\text{JSM} \end{aligned}$$

and we denote the "irregular indexing" by (J'JS) rather than (J,JS). The item corresponding to J and JS can be found by:

$$F(J'JS) = F(J+\text{JBJS}(JS))$$

In general, this is as computationally efficient as the conventional method of indexing. It requires additional storage, but generally not enough to present difficulty.

In the case of variables such as y or theta, JMS rows are bounded by JMS+1 interval boundaries, and the use of the pivot array is slightly different. For example, the larger of the two Y's bounding interval J in j-set JS is located by:

$$Y(J'JS) = Y(J+\text{JBJS}(JS)+JS)$$

Now, we define a new pivot array that will be a bit more indirect in definition, but much more useful:

$$\begin{aligned} \text{JBK}(1) &\equiv 0 \\ \text{JBK}(K+1) &\equiv \text{JBK}(K) + \text{JMBJS}(\text{JSET}(K)); \quad K=1,\dots,KM \end{aligned}$$

From this a function of J and K can be obtained immediately:

$$G(J'K) = G(J+\text{JBK}(K))$$

and it is convenient to note that JMS can be obtained in several ways, depending upon which data happen to be at hand:

$$\begin{aligned} \text{JMS} &= \text{JMBJS}(\text{JSET}(K)) \\ &= \text{JBJS}(\text{JSET}(K)+1) - \text{JBJS}(\text{JSET}(K)) \\ &= \text{JBK}(K+1) - \text{JBK}(K) \end{aligned}$$

and the overall number of rows is given by:

$$\begin{aligned} \text{JMKM} &= \sum_{K=1}^{KM} \text{JMBJS}(\text{JSET}(K)) \\ &= \text{JBK}(KM+1) \end{aligned}$$

This indexing scheme is the same method used in DOT 4 and DORT up to this point. The treatment of the I meshes follows the same plan, but it is a bit messier, since it is nested one layer deeper. We use input arrays:

$$\begin{aligned} \text{IMBIS}(\text{IS}) &\equiv \# \text{ of mesh cells in the ISth } i\text{-set} \\ \text{ISET}(J'K) &\equiv \# \text{ of the mesh set in the Jth row of the Kth plane} \end{aligned}$$

Once again, we can obtain:

$$\begin{aligned} \text{IS} &= \text{ISET}(J'K), \text{ the index of the } i\text{-set in row } J \text{ of plane } K \\ \text{IMS} &= \text{IMBIS}(\text{IS}), \text{ the length of } i\text{-set } \text{IS} \end{aligned}$$

Now, we define:

$$\begin{aligned} \text{IBIS}(1) &\equiv 0 \\ \text{IBIS}(\text{IS}+1) &\equiv \text{IBIS}(\text{IS}) + \text{IMBIS}(\text{IS}); \quad \text{IS}=1,\dots,\text{ISM} \end{aligned}$$

so that a function of I and IS can be located:

$$H(I'IS) = H(I+\text{IBIS}(\text{IS}))$$

For variables such as r or x, IMS cells are bounded by IMS+1 interval boundaries. For example, the larger of the two R's bounding interval I in i-set IS is located by:

$$R(I'IS) = R(I + \text{IBIS}(\text{IS}) + \text{IS})$$

Now, we define a new pivot array in terms of a linear variable, JK, that runs through each value of J for a plane, then through each plane in turn, plus a final terminating value; i.e.

$$JK=1,\dots,JMBS(JSET(1)),JMBS(JSET(1))+1,\dots,JKM,JKM+1$$

In terms of this variable, we now define:

$$\begin{aligned} IBJK(1) &\equiv 0 \\ IBJK(JK+1) &\equiv IBJK(JK) + IMBIS(ISET(JK)) \end{aligned}$$

From this, a function of I, J, and K that would be, with regular mesh:

$$P(I,J,K) = P(I+IM*((J-1)+JM*(K-1)))$$

becomes, with irregular indexing:

$$P(I'JK) = P(I+IBJK(J+JBK(K)))$$

We also define:

$$\begin{aligned} IJBK(1) &\equiv 0 \\ IJBK(K+1) &\equiv IJBK(K) + \sum_{J=1}^{JMBS(ISET(K))} IMBIS(ISET(J+JBK(K))); \quad K=1,\dots,KM \end{aligned}$$

This is needed for indexing things that vary by I and J, but not K:

$$Q(I'J) = Q(I+IBJK(J+JBK(K))-IJBK(K)); \quad K=\text{constant}$$

We also note that IMS can be obtained variously by:

$$\begin{aligned} IMS &= IMBIS(ISET(K)) \\ &= IBIS(ISET(J+JBK(K))+1) - IBIS(ISET(J+JBK(K))) \\ &= IBJK(J+JBK(K)+1) - IJBK(J+JBK(K)) \end{aligned}$$

and the overall number of mesh cells is:

$$\begin{aligned} IMJMKM &\equiv \sum_{K=1}^{KM} \sum_{J=1}^{JMBS(ISET(K))} IMBIS(ISET(J+JBK(K))) \\ &= IBJK(JMKM+1) \\ &= IJBK(K+1) \end{aligned}$$

APPENDIX E -- TORT BOUNDARY FLUX FILE FORMAT

```

-----
- name:      varbnd
-
- date:      03 march 1993
-
- purpose:   boundry source and associated interpretation data
-
- notes:     order of energy groups is by decreasing energy --
            neutrons, then photons.
-
- i is the first -dimension index.
- j is the second-dimension index.
- k is the third -dimension index.
- m is the direction index.
- zero is a word set to zero used in padding to full length.
-
- mult=1 if word length is 8 bytes; mult=2 if 4 bytes.
-
- when im.gt.0, the mesh is a regular (continuous) mesh with im cells
- in each row and jm rows in each plane. ism=jsm=ksm=1. ims=im.
- jms=jm. ima=im.
-
- when im.lt.0, the mesh is discontinuous. each plane contains
- jms rows, where jms=jmbjs(jset(k)). each row contains ims cells,
- where ims=imbis(iset(j'k)). (j'k) denotes j + sum of jms(kk)
- over kk=1,...,k-1. ima=iabs(im). ism is the number of i-sets.
- jms is the number of j-sets.
-
- when mm.gt.0, mm directions are used in the directional quadrature of
- flux in each energy group. msm=1. mms=mma.
-
- when mm.lt.0, the number of directions in the directional quadrature
- varies by group. mms=mbms(mset(ig)) is the number of directions
- used in group ig. mma=iabs(mm). msm is the number of direction
- sets.
-
- mmsdu(ig) is the larger of the number of downward or upward
- directions for the m-set used in ig. mmdnup is the largest mmsdu
- for any ig.
-
- special note: mmbms and mset are not on the output file produced
- by torsed and torset at this time.
-----

```

```

-----
- file structure:
-
- record type          present if
- -----
- file identification  always
- file label           always
- integer parameters   always
- indexing arrays      always
- real arrays          always
-
- .....do ig=1,igm
-
- .....do k=1,km

```



```

-   mmdnup          maximum number of directions down or up
-                   in any m-set
-   ism             number of i-sets
-   jsm             number of j-sets
-   imsism         sum of ims over is
-   jmsjsm         sum of jms over js
-
-   jmskm          sum of jms over km
-   mmsmsm         sum of mmbms over ms
-   msm            number of m-sets
-   ifbfxi         .eq.0 if i-boundary flux is included, else 1
-   ifbfj          .eq.0 if j-boundary flux is included, else 1
-
-   ifbfkx         .eq.0 if k-boundary flux is included, else 1
-   idum           array set to 0
-
-----

```

```

-----
- indexing arrays:
-   (imbis(is),is=1,ism), (jmbjs(js),js=1,jsm)
-   , ((iset(j'k),j=1,jms),k=1,km), (jset(k),k=1,km)
-   , (mmbms(ms),ms=1,msm), (mset(ig),ig=1,igm)
-
-   number of words= ism+jsm+jmskm+km+msm+igm
-
-   imbis          number of cells in i-set is
-   jmbjs          number of cells in j-set js
-   iset           i-set assigned to row j in plane k
-   jset           j-set assigned to plane k
-   mmbms          number of directions in m-set ms
-   mset           m-set assigned to energy group ig
-
-----

```

```

-----
- real arrays:
-
-   ((x(i,is),i=1,ims+1),is=1,ism), ((y(j,js),j=1,jms+1),js=1,jsm)
-   , (z(k),k=1,km+1), (ener(ig),ig=1,igm),emin,eneut, (dumrl(i),i=1,8)
-
-   number of words = imsism+ism+jmsjsm+jsm+km+1+igm+2+8
-
-   x              i-interval boundaries by i-set
-   y              j-interval boundaries by j-set
-   z              k-interval boundaries
-
-   ener           top energy boundary of group ig
-   emin           bottom energy boundary of group igm
-   eneut          bottom energy boundary of group neut
-                   (0 if neut=0)
-   dumrl         array set to 0.
-
-----

```

```

-----
- i-boundary source:
-
-   ((fio(m,j),m=1,mms),j=1,jms), (zero,l=1+mms*jms,mms*jm)
-
-   number of words = mms*jm

```

```

-      fio      i-boundary directional source
-
-----
- j-boundary source:
-      ((fjo(m,i),m=1,mms),i=1,ims), (zero,l=1+mms*ims,mms*ima)
-
-      number of words = mms*ima
-
-      fjo      j-boundary directional source
-
-----
- k-boundary source (top or bottom):
-      ((fko(m,i),m=1,mmsdu),i=1,ims), (zero,l=1+mmsdu*ims,mmsdu*ima)
-
-      number of words = mmsdu*ima
-
-      fko      k-boundary directional source, downward or upward
-               (for j.gt.jms, fko is filled with zero.)
-
-----

end

```

SECTION II

THE VISA CODE

1. INTRODUCTION

Upon request, the DORT code [rh88] produces a very large file containing directional fluxes for selected space mesh cells. It is the job of the Variable Input Source Assembly Code (VISA) to process those files into a form suitable for use in other codes. The task is complicated by several factors. First, it is necessary for DORT to write the data to the output file in the order in which they are generated, and this is not suitable for further use. Second, it is not practical for DORT to apply the results of the acceleration step on the last iteration. Because of this, the directional fluxes do not obey particle balance, and they do not match the scalar flux results. Third, the directional flux file generally contains more data than is required or can be conveniently processed by other codes, and VISA selects a more compact subset of that data. Finally, solving these difficulties requires the use of as many as three input files from two different codes, not convenient for the typical processing code.

VISA unscrambles this mess and prepares output in a format suitable for Monte Carlo adjoint folding in the Vehicle Code System (VCS) [rh74a,rh74b] or for continued 3-dimensional discrete ordinates calculations by TORSED, reported elsewhere in this document. VISA is not a highly polished code, and error checking, in particular, is fairly sparse. A number of options have been added over the years to take care of particular needs, and these make the input seem complicated to the uninitiated. Even so, the code has proven robust and reliable in two decades of application.

VISA was first reported in 1974, together with VCS. That reporting was quite brief, however, and this broader treatment is needed to meet modern requirements. The roots of VISA go back even farther, to the mid-1960's, and to an undocumented code called LIMBO. The identity of the authors of LIMBO is also undocumented, although it is suspected that early pioneers such as R.D.Rodgers, F.R.Mynatt, and/or M.L.Gritzner were responsible. Recognition for the preservation of LIMBO after the original authors moved on is due to J.V.Pace,III. Although the coding has been replaced and the function has been expanded, some of that basic idea survives in the modern product.

2. THEORETICAL INFORMATION

VISA uses input files containing directional flux and scalar flux data from DORT. First, VISA assembles the various components of the flux in each space cell into a suitably ordered array. Then, the weighted sum of the directional flux in each space cell over all directions is formed and compared with the corresponding scalar flux. The directional flux is renormalized so that the sums match.

If the DORT problem is started using a first collision source file such as that produced by the GRTUNCL code, that file is also required as input to VISA. (GRTUNCL is an undocumented code commonly distributed with DORT.) In addition to the source, the GRTUNCL file contains the magnitude of the uncollided flux for each cell. The magnitude of that flux is added to the DORT result in the direction nearest the ray extending from the GRTUNCL point source to the mesh cell. That result represents the total flux for the mesh cell.

It may be noted that the output of VISA is fluence, rather than flux, if the source used in GRTUNCL and/or DORT is a time integral. The term flux is used in the discussion as a convenience.

3. PROGRAMMER'S INFORMATION

3.1 Input, Output, and Common Blocks

The task of VISA, from a programmer's point of view, is to combine scattered information from two DORT files and one GRTUNCL file into a composite usable with the VCS code system or with the TORSED and TORT [rh91b,rh91a,rh87] codes. The DORT scalar flux file is produced in the VARFLM format described in Appendix D. The directional flux is produced in the "DOT Angular Flux Tape" format described in Appendix F. This format simulates a very old file format produced by DOT III [rh73] insofar as possible. The GRTUNCL file uses the VARSOR format used for distributed source input in DORT and described in Appendix E. The output file is prepared in the VISA2 format described in Appendix C.

Certain integer input parameters, specified in Appendix A, are required to control the execution. These data are read directly into common block COMVIS, described in Appendix B. In addition, a small array of real parameters and an array indicating which DORT radial intervals are to be included in the output are required, as listed in Appendix A. These are read into a single large "container" whose location is determined by calls to DORT support subroutines. The arrays are located in the container by the use of pointers as explained in the TORSED description. Additional input parameters and arrays are obtained from the input data files as indicated in Appendix B.

The general ordering of data arrays in the container is indicated by the order of the pointers in Appendix B. In certain special cases, data arrays may overlay each other if they are not used concurrently. The source listing is the final word on those details.

VISA uses several common arrays from DORT in order to communicate with DORT support subroutines. These blocks are:

COMIN -- general job status information
COMIO -- I/O unit status information
COMBLK -- reference address for the container

3.2 General Code Structure

The functioning of the code is illustrated by a summary of the main structural subroutines. In the order of occurrence, they are:

MAIN -- the main program reads the integer parameter data, obtains computer memory for the data container, sets pointer values for arrays in the container, and calls VISUS.

VISUS -- this routine calls DOPC to initialize the I/O process, reads the real parameter data, and couples arrays to the working subroutines according to the process chosen by the control parameters. At the conclusion, a final call to DOPC disposes all open data files properly.

WRVCS -- this routine reads the input files and merges the information as described earlier. The output is in a format suitable for use with VCS. For this application, the input flux is reorganized into records ordered by direction (M), then by axial interval (J). The records are written onto a random (direct) access scratch file according to radial interval (I), and then by energy group (IG). The records are then read in the order IG, then I, for writing to the final output. The subroutine is able to process input and output concurrently if the supporting DORT routines allow that.

WRTOR -- in this case, the input is processed as in WRVCS, but the output is produced for TORSED. The flux in each record is ordered by M, then I. The output records are ordered by J, then IG. No random access files are required.

UNTOR -- this special option allows a VISA file on unit NFLSV to be copied to unit NDATA. Data from a second VISA file on unit NUNCL can be added to the data from NFLSV. Data from each input file can be multiplied by a separate constant.

VCSTOR -- this allows data from a VISA file prepared for VCS and supplied on NFLSV to be re-sorted for use with TORSED and written to NDATA.

3.3 Supporting Service Routines

VISA draws extensively from the collection of service routines in the DORT distribution. A brief discussion of the routines is given in the TORSED description. In general, the source listing of each routine provides more detailed information as to the use and proper calling sequence. In some cases, these routines call other routines, and those have the same function as in DORT.

4. INSTALLATION AND ENVIRONMENT INFORMATION

4.1 Installation

The installation of VISA is accomplished as TORSED is installed, and no further user action is required. VISA is tested and demonstrated in each of the problem sets supplied with TORSED.

4.2 Data Files

Standard UNIX naming conventions are followed, in that the file associated with logical unit ?? is named fort.?. The standard input is assumed to be unit 5, and the standard output is unit 6. Those files are assumed to be opened by the system without explicit action by the program. The sequential data files containing the input from DORT and GRTUNCL, as well as the VISA output, may be any number between 1 and 80 except 5 and 6.

The code opens a scratch file on 91 if an output is being produced for VCS or if a VCS output is being converted to TORSED format. The size of the scratch file is:

variable	number	words	use
NT91	91	MM*NIP*NJP*IGM	sorting of VCS data

4.3 System Requirements

VISA is intended to be operable on any system on which DORT can be installed. The VISA routines are all written in FORTRAN 77, and they are in frequent use on both Cray mainframes and IBM workstations. The DORT routines make use of certain system-dependent features to provide special capability such as run-time memory allocation and timing data. Some C language is used in that area. The appropriate routines are provided in compatibility packages selected by the installation procedure. Generic routines are provided with DORT that should allow operation with somewhat reduced capability on systems for which no specific compatibility package exists.

The memory and CPU usage for VISA are quite nominal. A very large test problem produced a VCS-format output with MM=240, NIP=110, NJP=22, and IGM=212 on an IBM RS/6000 Model 320h workstation while using only 313,259 words of memory and 5 minutes of CPU+SYSTEM time. The elapsed time was 21 minutes. That file was later copied from an unformatted file to a formatted file in 53 minutes of CPU+SYSTEM time and 62 minutes of elapsed time.

5. USER INFORMATION

5.1 Input Requirements

The input data requirements are detailed in Appendix A. With the exception of the alphameric title, all data are input using the FIDO format also used by DORT and TORT. The user may refer to the corresponding documents for a specification of that format.

In the most general case, three input data files are also required on units specified in the parameter input:

NFLSV -- flux moment file produced by DORT on unit NTFOG.

NAFT -- "angular flux tape" produced by DORT on unit NTDIR.

NUNCL -- first collision source file produced by GRTUNCL on unit NPSO.

The output is written on:

NDATA -- output from VISA for use in VCS or TORSED.

The NAFT file DORT contains data for all I intervals and for all J between NJ1 and NJM. VISA restricts the output to NIP I intervals and $NJP = NPU - NPL + 1$ J intervals. Of course, NPL and NPU may be equal to NJ1 and NJM. The I intervals to be output are selected by entering NIP DORT I values in the input array IVAL.

If the NED parameter is set to a number of groups, the output flux for groups 1,...,NED will be edited. This produces an excessive amount of print for a large case. It is generally used only for testing. If NORM is set to 1, the normalization of the directional flux to agree with the scalar flux is defeated -- again, useful mostly for testing. If ISGRI is set to N, it is expected that NUNCL contains only N groups of data. This would be used, for example, if a coupled neutron-gamma problem were run using an NUNCL problem produced for neutrons only.

If DORT does not perform iterations on certain groups, those groups will be missing from the NAFT file. This can happen if the user decides to bypass calculation of certain groups, such as the neutron groups in a coupled problem, or if DORT decides not to iterate on some groups because of a 0 source in that group. If n initial groups of data are missing, setting NAFTI=n will produce a correct result. It may be noted that, if groups are missed after the first group of actual output, this condition is not repairable. If NTYPE.ge.10, NAFT is not required, and it may be 0.

The value of NTYPE controls the type of problem. Values of 0 or 1 produce the standard calculation as described above. Values of 10 or 11 copy a VCS or TORSED file previously produced by VISA from unit NFLSV to unit NDATA. A value of 21 produces a conversion from VCS to TORSED. If 30 or 31 is used, VCS or TORSED files on NFLSV

and NUNCL are added. Each is multiplied by a constant given in the real parameter input. The NEUI and NGAMX are needed only for special options discussed later. In all of the copy operations, the title from NFLSV replaces the title from the input stream.

If NFLSV, NAFT, NUNCL, and/or NDATA are set negative, the input or output corresponding to that file is expected to be formatted ASCII text, rather than an unformatted binary file. This option is useful largely in shipping data between unlike machines. The ASCII text can then be returned to unformatted using the copy option discussed above. The exact format can be deduced by comparing WANDR1 with the corresponding calling statements, although it is seldom necessary to do this.

5.2 VISA Without GRTUNCL Data or With GRTUNCL Data Only

If VISA is used to couple a reactor calculation to another calculation, for example, the DORT calculation may not have used GRTUNCL. In that case, NUNCL and ISGRI should be set to 0.

If ISGRI=0, the uncollided flux on NUNCL will be ignored, and only collided flux will be used. If ISGRI is set to negative, the uncollided flux will be used without the collided flux. These options are valuable for testing only.

5.3 Special Group Options

The parameters NEUI and NGAMX control some truly obscure options. If NTYPE=30 or 31, then NGAMX indicates the number of groups that are missing on NUNCL, but not on NFLSV. This can be used, for example, to add a gamma-only file to a coupled neutron-gamma file. If NTYPE=0 or 1, then NGAMX indicates the number of groups of 0 to be added between the last neutron group on NAFT and the first gamma group. In the latter case, the last neutron group must be indicated by NEUI.

If NEUI is set to $-1 \times$ the number of neutron groups, then the VISA output will contain only gamma groups.

5.4 Synthetic Flux

If NAFTI is set to $-1 \times N$, where N is a group number, the normal determination of flux is bypassed. Instead, fluxes in groups from 1 to N are set to 1.0, and the remainder of the fluxes are set to 0.0. This can be useful for testing.

5.5 Printed Output

The title and integer input parameters are edited first, followed by parameters from NAFT, some internally derived parameters, and then the real input parameters. If NED.gt.0,

the index of the direction matching the uncollided flux ray in each mesh cell is edited. In parallel columns, the input quadrature data are edited. This edit also includes the edit of the IVAL array, i.e. the I intervals selected for output, together with the R and Z interval midpoints chosen for the output.

As each group is assembled, a maximum scale convergence is edited, together with the first and last four words of each record. The scale convergence is the largest deviation of the renormalization factor from unity. If it is not small, on the order of the flux convergence in the DORT problem, there is a high likelihood of trouble with the input data. After this line, an edit of the total source moved to the output is given for groups indicated by the value of NED.

5.6 Error Messages

Certain conditions can produce error warnings and, if severe enough, a halt to execution. Those from MAIN and VISUS are self-explanatory. Certain other error messages can arise from the DORT service routines, and these have the same meaning as in DORT.

APPENDIX A -- VISA INPUT REQUIREMENTS

A.1 Title

A single line of identifying information (72 characters).

A.2 Control Parameter Input Block

1\$\$ -- Integer Parameters

nip = no. of i intervals in visa output
jpl = first j interval in visa output
jpu = last j interval in visa output
ned = edit ned source groups
norm = 0: normalize to scalar flux; 1: do not

isgri = no. of groups on nuncl (0: use collided flux only)
(negative: use uncollided only -- for testing)
nflsv = logical no. scalar flux input default=1
(old visa output if ntype.ge.10)
naft = logical no. directional flux input default=2
(may be 0 if ntype.ge.10)
nuncl = logical no. uncollided flux input default=3
(may be 0)
ndata = logical no. source output default=4

n5 = logical no. standard input default=5
n6 = logical no. standard output default=6
nj1 = first axial interval input; 0 implies=1
njm = last axial interval input; 0 implies=jm
nafti = no. groups missing at beginning of naft
(negative: first n groups=1, others 0 -- for testing)

ntype = 0/1: create vcs/torsed file; 10/11: copy vcs/torsed on nflsv;
21: vcs to torsed; 30/31: nuncl+nflsv
neui = last neutron group (reqd if ngamx.gt.0 and ntype=0,1)
(negative: delete neutron output groups)
ngamx = no. gamma groups added to group structure (ntype=0,1);
initial groups missing on nuncl (ntype=30,31)

[finish this array with 'e']

[terminate this block with 't']

A.3 Additional Array Block

2** -- Real Parameters

sh = height of point source (ntype.lt.10)
hsa = not used (enter 0)
xneut= nflsv multiplier (ntype=30or31, dflt=1)
xgam = nuncl multiplier (ntype=30or31, dflt=1)

[finish this array with 'e']

4\$\$ ival [nip entries] radial intervals to be included in output

[terminate this block with 't']

APPENDIX B -- COMMON BLOCK COMVIS

B.1 Parameter Input Pointers

la dummy array
lima length of common block
lfxt pointer for integer parameter input
lflt pointer for real parameter input
lend termination marker

B.2 Data Array Pointers

name	array set by	use
----	-----	---
lival	input data	dort i index corresponding to each visa output i
lwt	naft file	quadrature weight
lemu	"	cosine of direction with r axis
leta	"	cosine of direction with z axis
lrl	"	midpoint of radial interval
lz1	"	midpoint of axial interval
lphi	wrvcs, wrtor	azimuthal angle in plane perpendicular to r axis
ltheta	"	cosine of angle with r axis
liang	"	m of direction containing uncollided flux
laf	"	output flux
laflux	"	directional flux input from naft
lan2	"	scalar flux input from nflxsv
lunclf	"	uncollided flux input from nuncl
ldum	"	dummy array

B.3 Internal Working Parameters

name	set by	use
----	-----	---
nerr	main, visus	error flag
igi	naft file	no. of energy groups in DORT problem
igm	main	no. of energy groups processed by VISA
igp	"	igm+1
neut	naft file	no. of neutron groups in DORT problem
isgrp	main	no. of VISA source groups obtained from nuncl
naftm	"	no. of VISA source groups missing from naft
tdum	"	dummy array

B.4 Title Arrays Transferred To VISA Output File

name	set by	use
----	-----	---
title	input data	VISA job title
tdot	naft file	DORT job title

B.5 Internal Integers Parameters Transferred To VISA Output File

name	set by	use
----	-----	---
nm	naft file	no. of quadrature directions in DORT problem
im	"	no. of i intervals in DORT problem

jm	"	no. of j intervals in DORT problem
igo	main	no. of energy groups in VISA output
igop	"	igo+1
mmdn	wrvcs, wrtor	number of downward directions in DORT quadrature
njp	"	number of j intervals in VISA output
ish	visus	GRTUNCL point source height
ihsa	"	not used

B.6 Integer Input Parameters

The 18 integer input parameters listed in Appendix A are stored here, followed by:

name	set by	use
----	-----	---
igin		not used
igou		not used

B.7 Real Input Parameters

The 4 real input parameters listed in Appendix A are stored here.

B.8 File Input Scratch Array

name	set by	use
----	-----	---
xdum		dummy array
tinp	main	scratch array for input from naft or nflsv

APPENDIX C -- VISA OUTPUT FILE FORMAT

```

-----
-
- name:      visa2
-
- date:      17 march 1993
-
- purpose:   boundary flux for forward/adjoint folding in vcs or for
-            remapping by torsed and input to tort
-
- notes:     order of energy groups is by decreasing energy --
-            neutrons, then photons.
-
- i is the first -dimension index  (r axis).
- j is the second-dimension index  (z axis).
- m is the direction index
-
- output is either vcs format or torsed format.  the flux records
- depend upon this choice.
-
- if the source used to generate the input to visa is a time-integral,
- then the output is fluence, rather than flux.
-----

```

```

-----
- file structure:
-
- record type          present if
- -----
- job titles           always
- integer parameters   always
- integer array        always
- directional quadrature always
- space mesh           always
-
- .....do i=1,nip
- .....do ig=1,igm
- . . boundary directional flux      vcs format
- .....enddo
- .....enddo
-
- .....do ig=1,igm
- .....do j=1,njp
- . . boundary directional flux      torsed format
- .....enddo
- .....enddo
-----

```

```

-----
- job titles:
-
- (title(1),1=1,18), tdot(1),1=1,18)
-
- number of words = 36
-
- title      title of the visa job (a4 format)
- tdot      title of the dort job input to the visa job (a4 format)
-----

```

```

-----
- integer parameters:
-   titl(i)          title provided by user          - (a8)
-
-   mm,im,jm,igm,igp, mmdn,njp,ish,isha,nip, jpl,jpu
-   , (junk(1),l=1,11)
-
-   number of words = 23
-
-   mm              no. of quadrature directions in dort problem
-   im              no. of i interval midpoints in dort problem
-   jm              no. of j interval midpoints in dort problem
-   igo             no. of energy groups in visa output
-   igop           igo+1
-
-   mmdn           no. of downward directions in dort quadrature
-   njp            no. of j intervals in visa output
-   ish            grtuncl point source height
-   isha           not used
-   nip            no. of i intervals in visa output
-
-   jpl            index of first j interval in visa output
-   jpu            index of last j interval in visa output
-   junk           array of undefined integers to fill out length
-----

```

```

-----
- integer array:
-
-   (ival(i),i=1,nip)
-
-   number of words = nip
-
-   ival           dort i index corresponding to each visa output i
-----

```

```

-----
- directional quadrature:
-
-   (wt(m),m=1,mm), emu(m),m=1,mm), eta(m),m=1,mm)
-
-   number of words = 3*mm
-
-   wt             quadrature weight
-   emu            cosine of direction with r axis
-   eta            cosine of direction with z axis
-----

```

```

-----
- space mesh:
-
-   (r1(i),i=1,nip), (z1(j),j=1,njp)
-
-   number of words = nip+njp
-----

```

```
-      r1      radial mesh interval midpoint
-      z1      axial  mesh interval midpoint
-
-----
```

```
-----
- boundary directional flux (vcs format):
-
-      ((flux(m,j),m=1,mm),j=1,njp)
-
-      number of words = mm*njp
-
-      flux      boundary flux for forward/adjoint folding
-
-----
```

```
-----
- boundary directional flux (torsed format):
-
-      ((flux(m,i),m=1,mm),i=1,nip)
-
-      number of words = mm*nip
-
-      flux      boundary flux for torsed coupling to tort
-
-----
```

APPENDIX D -- DORT FLUX MOMENT FILE FORMAT

```

c*****
c              revised 10 nov 76
c
cf          varflm
ce          variable mesh flux moment data with boundary fluxes
c
c*****

```

```

cd          order of groups is by decreasing energy
cd          i is the first-dimension index
cd          j is the second-dimension index
cd          jm=1 for 1-dimensional geometry

```

```

c-----
cs          file structure
cs
cs          record type                                present if
cs          -----
cs          file identification                        always
cs          file label                                always
cs          file control                              always
cs          file integer parameters                  always
cs          file real    parameters                  always
cs
cs          ***** (repeat over all groups)
cs          *          flux moments                    always
cs          *          boundary directional flux        always
cs          *****
c
c-----

```

```

c-----
cr          file identification
c
cl          hname, (huse(i),i=1,2), ivers
c
cw          1+3*mult=number of words
c
cd          hname          hollerith file name - varflm - (a6)
cd          huse(i)        hollerith user identification - (a6)
cd          ivers          file version number
cd          mult           double precision parameter
cd                          1- a6 word is single word
cd                          2- a6 word is double precision word
c
c-----

```

```

c-----
c
cr          file label
c
cl          date,user,charge,case,time, (titl(i),i=1,12)
c
cw          17*mult=number of words
c
cd          date          as provided by timer option 4 - (a6)
cd          user          as provided by timer option 5 - (a6)
cd          charge        as provided by timer option 6 - (a6)
cd          case          as provided by timer option 7 - (a6)

```

```

cd    time          as provided by timer option 8 - (a6)      -
cd    titl(i)       title provided by user          - (a6)      -
c                                          -
c-----

```

```

c-----
cr          file control                                          -
c                                          -
cd    igm,neut,jm,lm,ima,mma,ism,imsism,isbt,iter,(idum(n),n=1,15) -
c                                          -
cw    25=number of words                                         -
c                                          -
cd    igm           number of energy groups                      -
cd    neut          last neutron group                            -
cd                   (igm if all neutrons, 0 if all gammas)      -
cd    jm           number of second-dimension (j) intervals      -
cd    lm           maximum length of moment expansion            -
cd    ima          maximum number of first-dimension intervals   -
cd    mma          number of boundary directions                 -
cd    ism          number of i-boundary sets                      -
cd    imsism       total number of i-intervals, all i-sets      -
cd    isbt         i-set for system boundaries                   -
cd    iter         outer iteration number at which flux was      -
cd                   written                                      -
cd    idum(i)      array set to 0                                 -
c                                          -
c-----

```

```

c-----
cr          file integer parameters                                -
c                                          -
cl    (lmbig(ig),ig=1,igm),                                       -
cl    *(imbis(is),is=1,ism),(iset(j),j=1,jm)                       -
c                                          -
cw    igm+ism+jm=number of words                                    -
c                                          -
cd    lmbig(ig)     length of moment expansion for group ig      -
cd    imbis(is)    number of intervals in iset is                 -
cd    iset(j)      i-set assigned to interval j                   -
c                                          -
c-----

```

```

c-----
cr          file real parameters                                  -
c                                          -
cl    (z(j),j=1,jm1),((r(i,is),i=1,im1),is=1,ism),               -
cl    *(ener(ig),ig=1,igm),emin,eneut,ev,devdk,effk,power,       -
cl    *(dumrl(i),i=1,13)                                           -
c                                          -
cw    jm+imsism+ism+igm+20=number of words                         -
c                                          -
cd    z(j)         j-interval boundaries                          -
cd    r(i,is)      i-interval boundaries for i-set i              -
cd    ener(ig)     top energy boundary of group ig                -
cd    emin         bottom energy boundary of group igm            -
cd    eneut        bottom energy boundary of group neut          -
cd                   (0 if neut=0)                                -
cd    ev           eigenvalue                                      -
cd    devdk        derivative of ev vs. effk                      -
cd    effk         effective multiplication factor                 -
cd    power        power (watts) to which flux is normalized     -

```

```

cd   dumrl          array set to 0      -
cd   jml            jm+1                -
cd   iml            imbis(is)+1        -
c                                         -
c-----

```

```

c-----
cr          flux moments                -
c                                         -
cl   ((flum(i,l),i=1,ims),l=1,lms)    -
c                                         -
cw   ims*lms=number of words          -
c                                         -
c   do 1 j=1,jm                        -
c 1 read(n) *list as above*          -
c                                         -
cd   flum          flux by interval and moment index -
cd   ims          imbis(is) for is corresponding to j -
cd   lms          lmbig(ig)           -
c                                         -
c-----

```

```

c-----
cr          boundary directional flux    -
c                                         -
cl   ((fio(m,j),m=1,mma),j=1,jm), ((fjo(m,i),m=1,mma),i=1,imb) -
c                                         -
cw   mma*(jm+imb)=number of words      -
c                                         -
cd   fio          directional flux outgoing by direction and -
cd                j-interval          -
cd   fjo          directional flux outgoing by direction and -
cd                i-interval. fjo=0 for a 1-d geometry      -
cd   imb          imbis(is) for is corresponding to isbt    -
c                                         -
c-----

```

end

APPENDIX E -- DORT SOURCE MOMENT FILE FORMAT

```

C*****
C          revised 04 jan 82
C
cf          varsor
ce          variable mesh source moment data
C
C*****

```

```

cd          order of groups is by decreasing energy
cd          i is the first-dimension index
cd          j is the second-dimension index
cd          jm=1 for 1-dimensional geometry
cd          if isop.gt.0, source is first-collision type

```

```

C-----
cs          file structure
cs
cs          record type                                present if
cs          -----
cs          file identification                        always
cs          file label                                always
cs          file control                              always
cs          file integer parameters                   always
cs
cs          ***** (repeat over all groups)
cs          *          source moments                  always
cs          *****
C
cs          ***** (repeat over all groups)
cs          *          scalar uncollided flux          isop.gt.0
cs          *****
C
C-----

```

```

C-----
cr          file identification
C
cl          hname, (huse(i), i=1,2), ivers
C
cw          1+3*mult=number of words
C
cd          hname          hollerith file name - varsor - (a6)
cd          huse(i)        hollerith user identification - (a6)
cd          ivers          file version number
cd          mult           double precision parameter
cd                          1- a6 word is single word
cd                          2- a6 word is double precision word
C
C-----

```

```

C-----
C
cr          file label
C
cl          date,user,charge,case,time, (titl(i), i=1,12)
C
cw          17*mult=number of words
C
cd          date          as provided by timer option 4 - (a6)

```

```

cd user as provided by timer option 5 - (a6) -
cd charge as provided by timer option 6 - (a6) -
cd case as provided by timer option 7 - (a6) -
cd time as provided by timer option 8 - (a6) -
cd titl(i) title provided by user - (a6) -
c -
c-----

```

```

c-----
cr file control -
c -
cd igm,neut,jm,lm,ima,mma,ism,imsism,isop,(idum(n),n=1,16) -
c -
cw 25=number of words -
c -
cd igm number of energy groups -
cd neut last neutron group -
cd (igm if all neutrons, 0 if all gammas) -
cd jm number of second-dimension (j) intervals -
cd lm maximum length of moment expansion -
cd ima maximum number of first-dimension intervals -
cd mma number of boundary directions -
cd ism number of i-boundary sets -
cd imsism total number of i-intervals, all i-sets -
cd isop uncollided flux flag -
cd 0 - no uncollided flux records present -
cd 1 - uncollided flux records present -
cd idum(i) array set to 0 -
c -
c-----

```

```

c-----
cr file integer parameters -
c -
cl (lmbig(ig),ig=1,igm), -
cl *(imbis(is),is=1,ism),(iset(j),j=1,jm) -
c -
cw igm+ism+jm=number of words -
c -
cd lmbig(ig) length of moment expansion for group ig -
cd imbis(is) number of intervals in iset is -
cd iset(j) i-set assigned to interval j -
c -
c-----

```

```

c-----
cr source moments -
c -
cl ((sorm(i,l),i=1,ims),l=1,lms) -
c -
cw ims*lms=number of words -
c -
c do 1 j=1,jm -
c 1 read(n) *list as above* -
c -
cd sorm source by interval and moment index -
cd ims imbis(is) for is corresponding to j -
cd lms lmbig(ig) -
c -
c-----

```

```
C-----  
cr          scalar uncollided flux  
C  
cl      (flum(1),i=1,ims)  
C  
cw      ims=number of words  
C  
C      do 1 j=1,jm  
C 1 read(n)  *list as above*  
C  
cd      flux          uncollided flux by interval  
C  
C-----  
  
end
```

APPENDIX F -- "DOT ANGULAR FLUX TAPE" FORMAT AS USED BY DORT

```

-----
-
- name:      "dot angular flux tape"
-
- date:      22 jun 1993
-
- purpose:   simulate traditional dot iii directional flux output file
-
- notes:     order of energy groups is by decreasing energy --
-            neutrons, then photons.
-
-   i is the first -dimension index, i=1,,,im
-   j is the second-dimension index, j=1,,,jm
-   m is the overall direction index, m=1,,,mm
-   ig is the energy group index, ig=1,igm
-
-   this format is not usable with discontinuous mesh, i.e. im.lt.0.
-
-   the simulation is not perfect in every detail; for example,
-   nisz and njszn did not appear in dot iii.
-
-   items listed as dummy have undefined values and should not be used.
-
-   fuller details may be found in the dot iv document, ornl-5851.
-----

```

- file structure:

```

-----
-
-   record type                present if
-   -----
-   control parameters         always
-   material by zone           always
-   zone number by mesh cell   always
-   fission spectrum           always
-   quadrature & input mesh boundaries always
-   quadrature mates           always
-
-   .....repeat for ig=1,igm
-   .   cross sections         always
-   .....end ig loop
-
-   .....repeat for ig=1,igm
-   .
-   .   .....repeat all j=jm,1,-1
-   .   . downward directional flux     always
-   .   .....end j loop
-   .
-   .   .....repeat all j=1,jm
-   .   . upward directional flux       always
-   .   .....end j loop
-   .
-   .....end ig loop
-
-   final radial mesh boundaries  always
-   "   axial   "   "   "
-----

```

- control parameters:

- idum1,iadj,isctma,mma,inggeom, izm,ima,jm,ktype,ev
- , evm,eps,nbcl,nbcr,nbct, nbc,inpfxm,mode,mtm,mixl
- , idum2,mtp,niszn,njszn,idum3, idum4,igm,iht,ihs,ihm
- , xnf,idum5,inpsrm,ifxmi,idum6, ifxmf,tmax,jdirf,jdirl
- , (dumary(i),i=1,18)

- number of words = 57

- idum1 dummy
- iadj forward/adjoint control
- isctma iabs(isctm); order of cross section expansion
- iabs(mm) no. of quadrature directions
- inggeom geometry option
- izm no. of material zones
- ima iabs(im); no. of radial mesh intervals
- jm no. of axial mesh intervals
- ktype calculation type
- ev eigenvalue
- evm eigenvalue modifier
- eps convergence criterion
- nbcl left boundary condition
- nbcr right " "
- nbct top " "
- nbc bottom " "
- inpfxm flux input option
- mode flux recursion option
- mtm total no. of cross section sets
- mixl length of mixing table
- idum2 dummy
- mtp no. of cross section sets from unit ntsig
- niszn no. of I super zones
- njszn no. of J super zones
- idum3 dummy
- idum4 dummy
- igm no. of energy groups
- iht position of total cross section
- ihs position of self-scatter cross section
- ihm no. of cross sections per group
- xnf source normalizer
- idum5 dummy
- inpsrm distributed source input control
- ifxmi initial inner iteration maximum per group
- idum6 dummy
- ifxmf final inner iteration maximum per group
- tmax problem time limit
- jdirf first j for directional flux output
- jdirl last " " " " "
- dumary dummy

- material by zone:

- (izmt(iz), iz=1, izm)

- number of words = izm

- izmt material number by zone
- izm total number of materials

- zone number by mesh cell:

- ((ijzn(i,j), i=1, im), j=1, jm)

- number of words = im*jm

- ijzn zone number for each material cell

- fission spectrum:

- (chi(ig), ig=1, igm)

- number of words = igm

- chi fission fraction per group

- quadrature & input mesh boundaries:

- (w(m), m=1, mm), (emu(m), m=1, mm), (eta(m), m=1, mm)
- * (rin(i), i=1, im+1), zin(j), j=1, jm+1)

- number of words = 3*mm + im + jm + 2

- w quadrature weight
- emu " cosine with r axis
- eta " " z "
- rin initial radial mesh boundaries
- zin " axial " "

- quadrature mates:

- (mtemu(m), m=1, mm), (mteta(m), m=1, mm), (mtlv1(m), m=1, mm)

- number of words = 3*mm

- mtemu quadrature emu mate
- mteta " eta mate 58

```

-   mtlvl           "           level mate
-
-----
- cross sections:
-
-   (sig(ih),mx=1,mtm)
-
-   number of words = mtm
-
-   sig             dummy array
-   mtm             number of materials
-
-----
- directional flux:
-
-   ((dirf(i,mx),i=1,im),mx=1,mm)
-
-   number of words = im*mm
-
-   dirf            cell-average directional flux
-
-----
- final radial boundaries:
-
-   (r(i),i=1,im+1)
-
-   number of words = im+1
-
-   r               final radial mesh boundaries
-
-----
- final axial boundaries:
-
-   (z(i),i=1,jm+1)
-
-   number of words = jm+1
-
-   z               final axial mesh boundaries
-
-----
end

```

SECTION III

REFERENCES

REFERENCES

- ch88b. R.L.Childs, W.A.Rhoades, and L.R.Williams, "Three-Dimensional Calculations of Neutron Streaming in the Beam Tubes of the ORNL HFIR Reactor," Proc. 7th Internat. Conf. on Radiation Shielding (Bournemouth, UK, September 1988).
- ch88a. R.L.Childs, F.B.K.Kam, R.E.Maerker, W.A.Rhoades, L.R.Williams, and C.A.Baldwin, "Neutron Dosimetry Analysis," Appendix E in "Evaluation of HFIR Pressure-Vessel Integrity Considering Radiation Embrittlement," R.D.Cheverton, J.G.Merkle, and R.K.Nanstad, Editors, ORNL/TM-10444 (April 1988).
- rh92b. W.A.Rhoades, R.L.Childs, and D.T.Ingersoll, "Radiation Exposure Inside Reinforced Concrete Buildings at Nagasaki," Health Physics 63, 5, 510-521 (November 1992).
- rh92a. W.A.Rhoades, R.A.Lillie, and M.B.Emmett, "Transmission Factors for the Penetration of Neutron and Photon Fluence into Wood-Frame Dwellings, 1990 (TF90)," ORNL/TM-12021 (May 1992).
- rh91b. W.A.Rhoades and R.L.Childs, "TORT -- Three-Dimensional Discrete Ordinates Neutron/Photon Transport Code," Proceedings of the Am. Nucl. Soc. Topical Mtg.: Advances in Mathematics, Computations, and Reactor Physics, pp. 30.3.1.1-30.3.1.4 (Pittsburgh, April 1991)
- rh91a. W.A.Rhoades and R.L.Childs, "TORT: A Three-Dimensional Discrete Ordinates Neutron/Photon Transport Code," Nucl. Sci. & Engr. 107, 4, pp. 397-398 (April 1991).
- rh89. W.A.Rhoades, R. L. Childs, and D. T. Ingersoll, "Radiation Exposure Inside Reinforced Concrete Buildings at Nagasaki," Oak Ridge National Laboratory, Oak Ridge, TN, ORNL/TM-10999 (May 1989).
- rh88. W.A.Rhoades and R.L.Childs, "The DORT Two-Dimensional Discrete Ordinates Transport Code," Nucl. Sci. & Engr. 99, 1, 88-89 (May 1988).
- rh87. W.A.Rhoades and R.L.Childs, "The TORT Three-Dimensional Discrete Ordinates Neutron/Photon Transport Code," ORNL-6268 (November 1987).
- rh74b. W.A.Rhoades, "Development of A Code System For Determining Radiation Protection of Armored Vehicles (The VCS Code)," ORNL-TM-4664 (October 1974).
- rh74a. W.A.Rhoades, M.B.Emmett, G.W.Morrison, J.V.Pace, III, and L.M.Petrie, "Vehicle Code System (VCS) User's Manual," ORNL-TM-4648 (August 1974).
- rh73. W.A.Rhoades and F.R.Mynatt, "The DOT III Two-Dimensional Discrete Ordinates Transport Code," ORNL-TM-4280 (September 1973).

th86. J.L.Thompson, M.B.Emmett, and H.L.Dodds, Jr., "Development and Evaluation of DOTTOR, A Computer Code to Couple Two-Dimensional to Three-Dimensional Discrete Ordinates Calculations," ORNL/TM-9919 (April 1986).

INTERNAL DISTRIBUTION

- | | |
|-----------------------|--------------------------------------|
| 1. B. R. Appleton | 24. L. R. Williams |
| 2. B. L. Broadhead | 25. B. A. Worley |
| 3-4. R. L. Childs | 26. EPMD Report Office |
| 5. D. T. Ingersoll | 27. Laboratory Records
Department |
| 6. R. A. Lillie | 28. Laboratory Records
ORNL-RC |
| 7. J. B. Manneschildt | 29. Document Reference
Section |
| 8. J. V. Pace, III | 30. Central Research
Library |
| 9. C. V. Parks | 31. ORNL Patent Section |
| 10. J. P. Renier | |
| 11-15. W. A. Rhoades | |
| 16-20. R. T. Santoro | |
| 21. C. O. Slater | |
| 23. R. C. Ward | |
| 24. R. M. Westfall | |

EXTERNAL DISTRIBUTION

- 32-36. D. A. Barnett, Knolls Atomic Power Laboratory, P. O. Box 1072, Schenectady, New York 12301
- 37. Prof. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street, Cambridge, MA 02138
- 38. C. A. Burre, Knolls Atomic Power Laboratory, P.O. Box 1072, Schenectady, New York 12301
- 39. Roy Castelli, Knolls Atomic Power Laboratory, P. O. Box 1072, Schenectady, New York 12301
- 40. Prof. Donald J. Dudziak, Depart, of Nuclear Engineering, 110B Burlington Engineering Labs, North Carolina State University, Raleigh, NC 27695-7909
- 41. Dr. James E. Leiss, Rt. 2, Box 142C, Broadway, VA 22815
- 42. Prof. Neville Moray, Dept. of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, IL 61801
- 43. Prof. Mary F. Wheeler, Dept. of Mathematics, Rice University, P.O. Box 1892, Houston, TX 77251
- 44. Office of the Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations, P.O. Box 2001, Oak Ridge, TN 37831
- 45-46. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, Tennessee 37830