NASA Reference Publication 1377

3DGRAPE/AL Users' Manual

Reese L. Sorenson and Stephen J. Alter

October 1995

National Aeronautics and
Space Administration

NASA Reference Publication 1377

3DGRAPE/AL Users' Manual

Reese L. Sorenson, Ames Research Center, Moffett Field, California
Stephen J. Alter, Lockheed Martin Engineering & Sciences, Hampton, Virginia

October 1995

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035-1000

TABLE OF CONTENTS

PAGE

ABSTRACT 1

INTRODUCTION 1

ACKNOWLEDGMENTS 3

OBTAINING AND INSTALLING THE PROGRAM 4
DISTRIBUTION 4
FILES COMPRISING THE GRID GENERATOR PROGRAM 6
PROGRAM FLOW ILLUSTRATED BY A CALL TREE 17

INPUT 25
THE FIRST TWO LINES 25
FILE10 -- CONTROL SCALARS FOR NEW START 26
The "run-comment" lines 27
The "number-of-blocks" line 27
The "iterations" lines 28
The "filename-11" line 32
The "filename-14" line 32
The "write-for-restart" line 34
The "omegpqr" line 35
The "quality-check" line 35
The "block-comment" line 36
The "dimension" line 37
The "handedness" line 37
The "polar-axis" line 39
The "freezeblock" line 40
The "face" line 40
The "norm/sect" line 44
The first type of "edges" line 44
The second type of "edges" line 45
The "read-in-fixed" line 47
The "plane-normal-to" lines 48
The "cylinder-about" lines 50
The "ellipsoid" line 52
The "collapsed-to-an-axis" lines 55
The "collapsed-to-a-point" lines 56
The "match-to-face" lines 57
The "freeze-at-restart" line 59
FILE11 – BODY DEFINITION ARRAYS 60
FILE12 – CELL HEIGHTS AND ANGLES AT BOUNDARY SURFACES 61
FILE13 – TO READ IN A GRID AND SMOOTH IT 63
FILE16 – CONTROL SCALARS FOR RE-START 63
FILE18 – INDICES OF SURFACES TO BE VIEWED 65
FILE19 – CONTROL SCALARS FOR SMOOTHING A GRID 67

OUTPUT 69

RUNNING THE GRAPHICAL USER INTERFACE 70

EXAMPLE CASES 72
THE BASIC BOX CASE 73
THE WING WITH FLAT PLATE EXTENSION CASE 80
THE HEMISPHERE-CYLINDER-CONE CASE 86
FIRST VARIATION ON THE BASIC BOX CASE -- THOMAS & MIDDLECOFF CLUSTERING TERMS 94
SECOND VARIATION ON THE BASIC BOX CASE -- LOCALLY OPTIMUM RELAXATION PARAMETER (Ω) 97

INPUT FILTERS 100
PREGRAPE/AL PROGRAM 100
F10FILTER PROGRAM 106

THEORETICAL DEVELOPMENT 107
POISSON EQUATIONS IN PHYSICAL SPACE 107
IMPROVED STEGER & SORENSON RHS TERMS 110
THOMAS & MIDDLECOFF CLUSTERING TERMS 115
OPTIMUM RELAXATION PARAMETER 116

POSTSCRIPT FILES 118

LIST OF TABLES

PAGE

Table 1. Directories Resulting from Unpacking the tar Tape 4
Table 2. Source File Naming Convention 5
Table 3. 3DGRAPE/AL Program Files (Files 1 - 159) 16
Table 4. List of Parameters 23
Table 5. Logical Unit Numbers Used in this Program 24
Table 6. The First Two Lines of Input Data 25
Table 7. Face Numbers and Indices 43
Table 8. List of Input Lines Used in File16 Input 65
Table 9. Color Codes 67
Table 10. GUI Control Buttons 71
Table 11. Screen Buttons in CONV HIST Mode 72
Table 12. Data Files for Example Cases (Files 160 - 178) 73
Table 13. PREGRAPE/AL Program and Data Files (Files 179 - 206) 101
Table 14. Description of PREGRAPE/AL input file 104
Table 15. Opposing Face Pairs 105
Table 16. F10FILTER Program and Data Files (Files 207 - 208) 107
Table 17. PostScript Files Comprising This Manual (Files 209 - 230) 119

ABSTRACT

This document is a users' manual for a new three-dimensional structured multiple-block volume grid generator called 3DGRAPE/AL.[1] It is a significantly improved version of the previously-released and widely-distributed program 3DGRAPE.[2,3] Many of those improvements are taken from the grid generator program 3DMAGGS.[4] It generates volume grids by iteratively solving the Poisson Equations in three-dimensions. The right-hand-side terms are designed so that user-specified grid cell heights and user-specified grid cell skewness near boundary surfaces result automatically, with little user intervention. Versatility was a high priority in this code's development, and as a result it can generate grids in almost any three-dimensional physical domain.

The code is written in Fortran-77. It can be installed as an ordinary batch program, and in that form it should run on almost any computer. Alternatively, on a Silicon Graphics Inc. (SGI) IRIS workstation it can be installed along with its simple graphical user interface (GUI). The GUI is also written in Fortran-77, and calls functions in the IRIS Graphics Library (IGL). With the GUI the user can watch selected grid surfaces converging to their final form as the elliptic solver iterates. For compiling on a CRAY supercomputer there is a vectorized batch version.

An introduction describing the improvements over the antecedent 3DGRAPE code is presented first. Then follows a chapter on the basic grid generator program itself, and comments on installing it. The input is then described in detail. After that is a description of the Graphical User Interface. Five example cases are shown next, with plots of the results. Following that is a chapter on two input filters: one which can change input for the antecedent 3DGRAPE program into input for this program, and the other which can prepare input data for this program from the output of GRIDGEN. Last is a treatment of the theory embodied in the code.

INTRODUCTION

The original program, 3DGRAPE, of which 3DGRAPE/AL is an updated version, is a batch-type program. This means that it reads in pre-defined input data, generates the grid, and writes it out. For those boundary surfaces which are of interest ("the body") it expects to read X,Y,Z coordinates of surface grid points which the user has pre-defined using other software. Other boundary surfaces of less interest ("the outer boundary") can be found by the program itself using simple analytic shapes. The grid can consist of multiple blocks, and the program is capable of finding its own internal block-to-block

---

[1]Sorenson, R. L. and Alter, S. J., "3DGRAPE/AL: The Ames-Langley Technology Upgrade," appearing in "Surface Modeling, Grid Generation, and Related Issues in Computational Fluid Dynamic (CFD) Solutions," NASA CP 3291, May 1995, pp. 447-462.

[2]Sorenson, R. L., "The 3DGRAPE Book: Theory, Users' Manual, Examples," NASA TM 102224, July 1989.

[3]Sorenson, R. L., "Three-Dimensional Zonal Grids About Arbitrary Shapes by Poisson's Equation," appearing in Sengupta, S., Häuser, J., Eiseman, P.R., and Taylor, C., eds., Numerical Grid Generation in Computational Fluid Mechanics, Pineridge Press Ltd., 1988.

[4]Alter, S. J., Weilmuenster, K. J., "The Three-Dimensional Multi-Block Advanced Grid Generation System (3DMAGGS)," NASA TM 108985, May 1993.

boundary surfaces. Volume grid points are found by numerically solving the Poisson equations. The Steger & Sorenson (S&S) right-hand-side (RHS) terms (or "forcing functions") in those equations are of a type which allows the user to choose the desired cell height on a read-in boundary, after which the program automatically finds the actual numerical values for the RHS terms which yield the desired cell heights. In the process the RHS terms attempt to give local near-orthogonality in the region of those same read-in surfaces. The cell heights the user requires may be of any magnitude (limited only by the precision of the computer), appropriate for both viscous and inviscid aerodynamic flow modeling. The input data is ordinary text, with required formatting. The output grid may be any of three formats, including the commonly used PLOT3D[5] formats.

The new program is called 3DGRAPE/AL. "GRAPE," as used herein, is an acronym standing for "Three-Dimensional GRids about Anything by Poisson's Equation." The "AL" signifies that the extension and improvement was performed by the two authors of this manual, one at NASA Ames Research Center and one at NASA Langley Research Center. All the features described above for the original program are preserved, and a significant suite of new features is added. Those new features are summarized below:

- Grid quality is enhanced by re-formulated control terms in the Poisson Equations. The user may specify arbitrary angles with which lines are to intersect boundaries, rather than that specification being limited to 90° everywhere. The treatment of sharp corners which transverse boundary surfaces (e.g., a grid wrapping around an airplane fuselage which has a strake) is improved using this capability.
- Another improvement to grid quality is the addition of Thomas & Middlecoff[6] (T&M) clustering terms for cases where all six faces of a block are read-in. The user can choose either the Steger & Sorenson terms (as in the original code and improved as described above), the Thomas & Middlecoff type terms, or a blending between the two which gives good cell-size and skewness control at both the boundaries and the interior.
- Grid quality is evaluated by computing and printing maxima, minima, medians, and averages of cell heights and non-orthogonality, at boundaries and in the interiors of the blocks of the finished grid.
- Initialization is improved by Trans-Finite Interpolation[7] (for cases with six fixed boundary surfaces). In some cases grids initialized thusly can serve as the final grid, in others this improved initialization speeds convergence.
- Erlich's Ad Hoc Method[8] for computing locally optimum relaxation parameters is available for the code's SOR solver. This also can speed convergence.
- When installed on CRAY computers the code is vectorized in all three coordinate directions, allowing the longest possible vector length in each block. This, also, speeds convergence.

---

[5]Walatka, P. P., Buning, P. G., and Elson, P. A., "PLOT3D User's Manual," NASA TM 101067, July 1992.

[6]Thomas, P. D., Middlecoff, J. F., "Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations," AIAA Journal, vol 18, pp. 652-656, June 1979.

[7]Soni, B. K., "Two- and Three-Dimensional Grid Generation for Internal Flow Applications of Computational Fluid Ddynamics," AIAA 85-1526, 1985.

[8]Erlich, L. W., "An Ad Hoc SOR Method, " Journal of Computational Physics, vol. 44, pp. 31-45, March 1981.

- The grid generation iteration schedule can be divided into parts. Parameters which effect convergence (such as relaxation rates), as well as the type of clustering terms used and their associated decay rates, are adjustable with each part. Intermediate solutions and restart files can be written after each part. Thus, in practical operation, as much can sometimes be accomplished in one run with this program as in multiple runs with other grid generators.
- An input filter called PREGRAPE/AL is supplied as a companion program. It inputs the output from the GRIDGEN[9] code, which contains blocking strategy and surface grids, and turns that into input for 3DGRAPE/AL.
- Another input filter, called F10FILTER which reads input designed for the earlier 3DGRAPE program, and re-formats it for use in the new 3DGRAPE/AL program.
- Required cell heights and skewness at read-in surfaces can be specified by the user at each point from a file.
- A complete grid generated elsewhere can be read-in, and the elliptic solver can be run a few steps to smooth the grid.
- The program has much more extensive error-checking, of both input data and the process following.
- A simple Graphical User Interface, coded in Fortran-77 and calling the IRIS Graphics Library, allows the user to watch selected grid surfaces while the grid solver is iterating. The user can also pause anytime during the iterative process and plot convergence histories. A full suite of transforms and other features is included.

ACKNOWLEDGMENTS

The authors are grateful to Professor Joe Thompson for his pioneering work in elliptic grid generation, and for his gracious and encouraging example. Dr. Jeffrey Hultquist, formerly of NASA Ames Research Center, provided invaluable help with the graphics; were it not for his patience, generosity, and expertise the GUI would not have been possible. Dr. Jin Chou of Computer Science Corp. contributed expert assistance with vector analysis. Dr. Jamshid S. Abolhassani of Computer Sciences Corporation provided various valuable insights and explanations. Mr. William Kleb of NASA Langley Research Center assisted in the formulation of LARCS and other interpolation issues. The second author gratefully acknowledges support from contract NAS1-19000.

This work is dedicated to the memory of Joseph L. Steger: scientist, mentor, teacher, and friend.

[9]Steinbrenner, J. P., Chawner, J. R., and Fouts, C. L., The GRIDGEN 3D Multiple Block Grid Generation System," Wright Research and Development Center Report WRDC TR90-33022, October 1989.

OBTAINING AND INSTALLING THE PROGRAM

DISTRIBUTION

It is intended that the code will be available from NASA's clearinghouse for computer programs, the Computer Software Management and Information Center (COSMIC), located at:

COSMIC
University of Georgia
382 East Broad Street
Athens, GA 30602

Phone: (404) 542-3265

Internet: service@cosmic.uga.edu

It is expected that the code will be distributed on a UNIX tar tape. How COSMIC will chose to structure the files on that tape -- their order and the directory structure into which they are put -- is unknown as of this writing. However, as the tar tape was distributed by the authors, unpacking it caused the creation of seven directories:

Directory:	Contents:
3dgdrape	Program files, makefiles, and header files making up the 3DGRAPE/AL grid generator code
box	Input and output data files for the three sample data cases using the "wavy-sided box" boundary shape
wing	Input and output data files for the "wing-and-flat-plate" sample data case
hcc	Input and output data files for the "hemisphere-cylinder-cone" sample data case
pre	Program files, makefiles, header files, and sample data files for the PREGRAPE/AL input filter
f10filter	Program file and sample data file for the F10FILTER input filter
ps	Compressed PostScript files containing the text and figures which are this manual

Table 1. Directories Resulting from Unpacking the tar Tape

Two hundred and thirty files are distributed among the directories. Every main program, subroutine, function, makefile, datafile, etc., is in its own file. Those 230 files are described in several tables appearing through out this manual. Files 1 through 159 are files associated with the 3DGRAPE/AL grid generator code, including the GUI and makefiles. Files 160 through 178 are files which make up the five example cases which are used to exercise the grid generator code. Files 179 through 206 constitute the input filter PREGRAPE/AL, including a makefile and a test case. Files number 207 and 208 are source code and a test case for the input filter F10FILTER. Files 209 through 230 are compressed PostScript files which comprise this manual.

Because the program was developed on a computer having a UNIX operating system the developer has made use of the C-Pre-Processor utility, which offers functionality similar to that of the Update utility on earlier CDC and CRAY computers. This means, simply, that a utility has been employed wherein parameter statements and common blocks are stored once, in separate files called "header files," and then automatically inserted into the code, wherever needed, by #include statements. Thus, when a dimension size or a common block must be changed, that change is made in only one place. This reduces both effort and errors. Users employing the code on other operating systems not having this utility should simply insert copies of the header files wherever they are required.

Another artifice, also deriving from the UNIX operating system on which this code was developed, is the makefiles. They constitute explicit instructions of how the code is to be compiled and linked. There are six makefiles supplied with the code, for compiling and linking it in UNIX operating systems running on

- single-processor SGI workstations,
- multiple-processor SGI workstations,
- CRAY supercomputers,
- SUN workstations,
- IBM workstations, and
- H-P workstations.

On SGI workstations the program can be compiled and linked either with or without the graphics package; on the other four types of machine the supplied makefiles will compile and link it as a batch program only, without any graphics. As of this writing it is said that the IGL has been licensed to IBM, and so the program with its graphics package might run on IBM machines as well as SGIs, but this has not been tested. In its batch version the code should run on anything with a Fortran-77 compiler.

To assist the user in differentiating between the various parts and options of this software package a naming convention has been used for the source files:

Source files having names ending in:	**Contain:**
Just plain ".f"	The basic batch version of 3DGRAPE/AL. They are used on all the computers listed above. Program "main.f" is an example.
"_v.f"	Versions of the solve subroutines which vectorize on the CRAY. File "solve_v.f" is an example.
"_m.f"	Versions of the solve subroutines which are optimized for use on multiple-processor SGI workstations. File "solve_m.f" is an example.
"_g.f"	The graphics package. They are used only on SGI workstations. File "plotit_g.f" is an example.
"_p.f"	PREGRAPE/AL. Main program "pregrapeal_p.f" is an example.

Table 2. Source File Naming Convention

FILES COMPRISING THE GRID GENERATOR PROGRAM

The table below gives a list of the files containing the 3DGRAPE/AL code and other files necessary to compile and link it. It gives a file number, the file name, comments on that program unit, and a notation of how this program unit is different from its antecedent in the earlier version of 3DGRAPE. All the files listed below will be found in the subdirectory "3dgrape".

File number:	File name:	Purpose of, and assorted observations on, the file:	How changed:
1	main.f	The main program. The same in all versions.	Much simplified
2	axbd.f	Applies the collapsed-to-axis boundary treatment by calling subroutine axsub. Does the indexing.	
3	axinit.f	Initializes the X,Y,Z for points on an axis.	
4	axsub.f	Actually extrapolates a line to an axis.	
5	banner.f	Writes the 3DGRAPE/AL "banner" onto the "printout" file.	New subroutine, code taken from subroutine input
6	boundary.f	Applies the boundary conditions in each iteration. New boundary treatment -- freeze-at-restart. The user might want the floating boundaries to stop floating at restart.	New boundary treatment added
7	buglist.f	Collect brief notes concerning bugs found and fixed. Set bugfix level number for printout.	New subroutine
8	checkco.f	Consider whether the boundary treatments specified for each face cause the corner points to be treated not at all, once, or more than once.	New subroutine
9	checked.f	Consider whether the boundary treatments specified for each face cause the edges to be treated not at all, once, or more than once.	New subroutine
10	checkhow.f	Go through each edge, recording how each point on each edge is treated.	New subroutine
11	checks.f	Check each edge point and each corner point to see if the boundary treatments specified for each face cause those points to be treated not at all, once, or more than once. This subroutine calls checkco, checked, checkhow.	New subroutine
12	chkmat.f	Checks the match-to-face input data for consistency.	More robust

13	coarse.f	Subroutine coarse cycles through the coarse parts in the iteration schedule, and then interpolates from coarse to fine. Batch version.	New subroutine, code taken from main program.
14	cylbd.f	Does the indexing and calls cylsub to apply the cylinder-about boundary treatment. As with axbd and axsub, this does the indexing while cylsub actually does the work.	
15	cylinit.f	Initializes the X,Y,Z for points on a cylinder.	
16	cylsub.f	Actually projects a line onto a cylinder.	
17	docoarse.f	A little logical function which tells us whether or not there are any coarse parts in the iteration schedule, and if the requisite conditions are satisfied.	New subroutine
18	edge.f	Given a function of one independent variable, discontinuous and double valued (e.g., the tangent at pi/2), this function finds a working value of the function at the point of discontinuity by extrapolating to that point from both sides, and averaging those two values. It is part of the generalized angle treatment.	New subroutine
19	elipbd.f	Does the indexing and calls elipsub to apply the ellipsoid boundary treatment. As with axbd and axsub, this does the indexing while elipsub actually does the work.	
20	elipinit.f	Initializes the X,Y,Z for points on a cylinder. Completely re-written to initialize those points as at the intersections of lines of latitude and longitude on a globe. Allows polar axis to be any of the 3 coordinate axes, and either index to go in either direction. Removes the awkward restriction about being only an even-numbered face, with read-in-fixed face opposite it.	Much improved
21	elipsub.f	Actually project a line onto the ellipsoid. Completely re-written to have the capability to truly project along a local normal to the ellipsoid, whereas before we could only project from the origin to the ellipsoid. This solves a problem, seen in the earlier code, wherein these boundary points were "stiff," i.e., they refused to move much with iteration.	Much improved.

22	fine.f	Subroutine fine cycles through the fine parts in the iteration schedule. As with subroutine coarse, above, it comes in a batch and a graphical version. This is the bvatch version.	New subroutine, code taken from main program.
23	fixinit.f	Reads the X,Y,Z for read-in-fixed boundary treatments. Now has the ability to use 12-column fields or 20- column fields in file11, at the user's choice.	New format available.
24	frezinit.f	Initializes things for the new frozen-at-restart boundary treatment.	New subroutine
25	getang.f	Part of the grid quality package. A function to find the angle between two vectors.	New subroutine
26 27 28	getdsi12.f getdsi34.f getdsi56.f	Record appropriate values for the cell height and cell skewness.	New subroutines, code taken from poif... subroutines.
29	getedges.f	Reads input data for sharp corners cutting across faces.	New subroutine, replaces subroutine light
30	getmedan.f	Given an un-sorted list of numbers, find the median entry. Part of the grid quality package.	New subroutine
31	getsmoo.f	Reads input from files 13 and 19 for the case wherein a grid is read in and smoothed.	New subroutine
32	getstdev.f	Given a list of numbers calculate their standard deviation. Part of the grid quality package.	New subroutine
33	init1d.f	Applies Vinokur's two-ended stretching algorithm to do stretched 1-D initialization between opposing faces of the users choice.	New subroutine, replaces subroutine newinit
34	initcoms.f	Initializes all the common variables, mostly to zero.	New subroutine
35	input10.f	Reads input from file10, and calls other subroutines which do the same. Initializes some variables.	Formerly called subroutine input.
36	input16.f	Reads input from file16, in the case of a restart, and calls other subroutines which do the same. Initializes some variables.	New subroutine, code taken from subroutine restart

37	input19.f	Reads input from file19, and calls other subroutines which do the same. This is the case of reading in an already-generated grid and smoothing it a little. Initializes some variables.	New subroutine
38	interp.f	Calls subroutines interp1, interp2, and interp3 to interpolate X,Y,Z and P,Q,R from coarse to fine.	
39 40 41	interp1.f interp2.f interp3.f	Interpolate X,Y,Z and P,Q,R from coarse to fine.	
42	jiggle.f	Use a random number generator to move the interior points around just a little, to prevent blow-up on the first iteration for certain kinds of initial conditions.	
43	larcs.f	Smoothes a surface.	New to this code
44	lower.f	Converts all incoming text to lower case, to make it easier to test on that text.	
45	makerhs.f	Note how the RHS are to be calculated, and calculate them.	New subroutine
46	matbd.f	Applies the match-to-face boundary treatment.	
47	matinit.f	Reads input and initializes points for the match-to-face boundary treatment.	
48	normst.f	Reads in and processes cell height data for specification of cell heights by stations.	
49	outparts.f	Output grid files and restart files after each of the individual parts in the iteration schedule, if appropriate.	New subroutine
50	output.f	Does the various types of output after the grid is generated.	New subroutine, code taken from main program and modified.
51	plabd.f	Applies the plane-normal-to boundary treatment by doing the indexing and calling subroutine plasub.	
52	plainit.f	Reads input and initializes points on faces having the plane-normal-to boundary treatment.	More robust
53	plasub.f	Actually projects a line to a plane.	More robust
54	pntinit.f	Reads data and initializes a face to be collapsed to a point.	

55 56 57	poif12.f poif34.f poif56.f	Calculates the terms in the Steger & Sorenson (S&S-type) RHS terms which are invariant with respect to computational time.	Formerly six subroutines, now three. Improved by adding generalized angle control.
58 59 60	q2d12rel.f q2d34rel.f q2d56rel.f	Compute cell height, and angles between lines intersecting the surface and that surface, relative to what was locally specified. Part of the grid quality package.	New subroutines
61 62 63	qual2d12.f qual2d34.f qual2d56.f	Compute cell height, and angles between lines intersecting the surface and that surface in absolute terms. Part of the grid quality package.	New subroutines
64	quality.f	The driver for the grid quality package. Calls the other subroutines, and prints out the answers.	New subroutine
65	qualorth.f	Computes measures of non-orthogonality at each point in the interior of a block. Part of the grid quality package.	New subroutine
66 67 68	qualsrj.f qualsrk.f qualsrl.f	Compute stretching ratios in the indicated coordinate directions in the interior of a block. Part of the grid quality package.	New subroutines
69	readangs.f	Read, from file12, the angle between the line intersecting the surface and each of the two surface coordinate lines.	New subroutine
70	readhi.f	Read, from file12, cell heights at every point on a face.	New subroutine
71	restart.f	Reads or writes data for restart.	Major re-write.
72 73 74	rhsf12.f rhsf34.f rhsf56.f	The S&S-type RHS terms are linear functions of a second derivative near the surface, with the constant coefficients calculated by the poif... routines. These subroutines calculate that derivative at the current time step, and then re-compute the S&S-type RHS terms.	Formerly six subroutines, now three.
75 76	sinhinv.f sininv.f	Solve for x in the equations y=(sin(x))/x and y=(sinh(x))/x, as required by Vinokur's stretching function. As no analytic solution is known, we must use approximations.	New to this code, but re-named versions of functions of indeterminat e age and unknown authorship.

77	solve.f	Apply the SOR iterative scheme to the grid generation equations and get the grid. Used on single-processor SGI workstations, and on SUN, IBM, and H-P workstations.	Much modified.
78	solve_v.f	Apply the SOR iterative scheme to the grid generation equations and get the grid. Used on CRAY and multiple-processor SGI workstations. It doesn't actually iterate the equations; it decides in which direction each block should be vectorized and then calls the subroutines immediately below.	New subroutine
79 80 81	solvej_v.f solvek_v.f solvel_v.f	Actually apply the SOR iterative scheme to the grid generation equations and get the grid. These subroutines are vectorized in their respective coordinate directions on the CRAY.	New subroutines, but patterned on the old solve
82 83 84	solvej_m.f solvek_m.f solvel_m.f	Actually apply the SOR iterative scheme to the grid generation equations and get the grid. These subroutines are optimized for use on multiple-processor SGI workstations.	New subroutines, but patterned on the old solve
85	sphbox.f	Convert the outermost 3 surfaces on all 6 sides of any block into or out of spherical coordinates.	Slightly modified.
86	sphchk.f	When going in and out of spherical coordinates there is a problem. The phi angle is the output from an arctan function, which is multiple-valued. The phi can be on different branches of the function. This subroutine attempts to correct that, and put them back on the same branch.	
87	sphio.f	Take any 3-D region, mapping into a rectangular solid in the computational domain, and convert it into or out of spherical coordinates.	Much modified to be more robust
88	sphpre.f	In spherical coordinates the angles, in radians, are going to be on the order of 1. But the radii can be on any order. These different scales can lead to numerical problems. The solution is to scale things, generate the grid, then unscale. This subroutine prepares those scale factors.	
89	sphsub.f	Called by sphchk which does the indexing. This actually does the work.	

90	startup.f	This gets the code ready to iterate. It calls the appropriate input subroutine, prepares those terms in the RHS which are fixed for all computational time, initializes the interiors of the blocks, takes the grid into and out of spherical coordinates if appropriate, and first calls the graphics if appropriate.	New subroutine. Code taken from main and much modified.
91	stretch.f	Implements Vinokur's stretching function to give a normalized tabulated data from 0 to 1.	New to this code, but re-named version of a subroutine of indeterminat e age and unknown authorship.
92	tfi2d.f	Performs 2-D Trans Finite Interpolation. Used in preparing the T&M-type RHS terms.	New to this code.
93	tfi3d.f	Performs 3-D Trans Finite Interpolation. Used in initializing the interiors of the blocks, in the case of all six sides of the block read-in-fixed.	New to this code.
94	tm.f	Calculate the T&M-type RHS terms.	New to this code
95	tweakpqr.f	For the case of RHS terms being S&S-type blended with T&M-type, in each iteration we must take the p1, q1, and r1 terms and subtract the T&M-type terms at the wall, compute the P, Q, and R terms at each point, update the X,Y,Z, and restore the p1, q1, and r1 by adding the T&M-type back in. This subroutine adds and subtracts the T&M-type terms at the wall.	New subroutine
96	writeit.f	Writes the grid solution file, file14.	
97	xferpqr.f	Initialize the S&S-type RHS terms to the T&M-type values if appropriate.	New subroutine
98	plotit_g.f	The driver for the graphics package. The call to this, and some common blocks, are the only interface between the batch part of 3DGRAPE/AL and this graphics package.	New subroutine
99	adjust_g.f	The grid surfaces the user wants to plot typically contain some index values not present in the coarse solution. So if we are plotting a coarse solution we must modify the requested index values to contain only coarse points. Do so here.	New subroutine

100	axislims_g.f	Given the minimum and maximum values of data represented by an axis, find "nice round numbers" for the minimum, maximum, and ticmark intervals used in plotting the axis.	New subroutine
101	byebye_g.f	Terminate graphical activity and exit the code.	New subroutine
102	coarse_g.f	Subroutine coarse cycles through the coarse parts in the iteration schedule, and then interpolates from coarse to fine. Graphics version. It calls plotit.	New subroutine, code taken from main program.
103	cross_g.f	The when plotting the grid we have the ability to zoom in and out. But that is done on whatever is at the exact center of the window. To put a region of interest at the center of the window, we need to know where the center is. This subroutine puts a multi-colored cross at the exact center of the window.	New subroutine
104	datlin_g.f	Draw the actual convergence history lines on the plot.	New Subroutine
105	dobut23_g.f	The graphics has a screen button marked "exit". But exiting can be complicated, with confirmation and all. This subroutine processes that button hit.	New subroutine
106	dobut82_g.f	Manage button hits and create the window for convergence history plots.	New subroutine
107 108	drawinsa_g.f drawinsi_g.f	Draw the three control windows on the right side of the screen, in their active and inactive modes, respectively.	New subroutine
109	drawtris_g.f	Draw the little green triangles which indicate the speed settings.	New subroutine
110 111	drawxax_g.f drawyax_g.f	Draw the axes on the convergence history plots.	New subroutines
112	findlegy_g.f	Find the vertical location of the legend so that it covers the fewest data points.	New subroutine
113	fine_g.f	Subroutine fine cycles through the fine parts in the iteration schedule. As with subroutine coarse, above, it comes in a batch and a graphical version. This is the graphical version. It calls plotit.	New subroutine, code taken from main program.
114	getlims_g.f	Get the minima and maxima of the convergence history data to be plotted.	New subroutine
115	grstart_g.f	Access the grid through the common blocks, and make the grid plot objects upon first entry to the graphics package.	New subroutine

116	grstart2_g.f	Access the grid through the common blocks, and re-make the grid plot objects upon subsequent entry.	New subroutine
117	histpl_g.f	Plot a convergence history.	New subroutine
118	kulur_g.f	Select colors by number.	New subroutine
119	legend_g.f	Plot the legend on the convergence history plots.	
120	lenstr_g.f	Find the length of a character string.	
121	limitit_g.f	Impose a limit on how large or small the absolute value of a number may be.	
122	makecobj_g.f	Make a plot object of the control window in every possible button state.	New subroutine
123	maketobj_g.f	Make a plot object of the transform window in every possible button state.	New subroutine
124	maketri_g.f	Make a plot object for the little green triangles.	New subroutine
125	makewin5_g.f	Make the window for plotting the convergence history plots.	New subroutine
126	makewins_g.f	Actually make the three small windows on the right.	New subroutine
127	makextrp_g.f	Make the exit trap plot object for the control window.	New subroutine
128	mkhistob_g.f	Make the screen button objects for the history plot window.	New subroutine
129	mkvsobj_g.f	Make the view selection objects for the view selection window.	New subroutine
130	mmacts_g.f	Interpret the actions of the middle mouse button.	New subroutine
131	movetri_g.f	Interpret the mouse movement to determine the speed settings.	New subroutine
132	onbut_g.f	A logical function. Is the mouse on a specific button?	New subroutine
133	pauz_g.f	Wait for a mouse hit on a screen button in the convergence history plot.	New subroutine
134	pline_g.f	Draw a line between two given points.	New subroutine
135	plstart_g.f	Get ready to plot by calling all the subroutines which make plot objects and open windows.	New subroutine
136	prepdata_g.f	Prepare history data for plotting.	New subroutine

137	pvwds_g.f	Draw words vertically.	New subroutine
138	transfm_g.f	Do the transforms -- translation or rotation.	New subroutine
139	wfiloop_g.f	Loop while waiting for a mouse interrupt.	New subroutine
140	zbufit_g.f	Put the program in Z-buffer mode, and re-draw the scene.	New subroutine
141	makefile.cray	A UNIX makefile to compile and link the code, in the vectorized version, on a CRAY.	New makefile
142	makefile.hp	A UNIX makefile to compile and link the code on an H-P workstation.	New makefile
143	makefile.ibm	A UNIX makefile to compile and link the code on an IBM workstation.	New makefile
144	makefile.sgi	A UNIX makefile to compile and link the code, in either batch or graphical version, on a single-processor SGI workstation.	New makefile
145	makefile.sgi_m	A UNIX makefile to compile and link the code, in either batch or graphical version, on a multiple-processor SGI workstation.	New makefile
146	makefile.sun	A UNIX makefile to compile and link the code on a SUN workstation.	New makefile
147	blend.h	Header file containing a common statement containing arrays containing data used in blending between the S&S-type and T&M-type RHS terms	New header file
148	etc.h	Header file containing a common statement containing various assorted scalar variables and small arrays which don't logically fit anywhere else.	New header file
149	faces1.h	Header file containing a common statement containing arrays containing gammas (see the transformed Poission equations), partial derivatives on the faces, coefficient terms in the S&S-type RHS terms, etc.	New header file
150	faces2.h	Header file containing a common statement containing arrays containing various data per block	New header file
151	files.h	Header file containing a common statement containing character variables which are the various filenames used in the code for reading and writing	New header file
152	history.h	Header file containing a common statement containing arrays containing data used in the convergence history	New header file

153	komment.h	Header file containing a common statement containing various character variables (Fortran-77 frowns on having these in the same common blocks as other types of variables)	New header file
154	limits.h	Header file containing a common statement containing arrays containing the limits and increments of the indices, for the various blocks	New header file
155	matches.h	Header file containing a common statement containing arrays holding various data per block	New header file
156	params.h	Header file containing a parameter statement giving common array dimensions	New header file
157	plcoms.h	Header file containing common statements containing various scalar variables and arrays used by the plotting package	New header file
158	tmcntrl.h	Header file containing a common statement containing arrays containing the T&M-type RHS terms	New header file
159	xyzcom.h	Header file containing a common statement containing X, Y, and Z arrays	New header file

Table 3. 3DGRAPE/AL Program Files (Files 1 - 159)

Readers familiar with the earlier code will note that in several cases a large and unwieldy subroutine has been broken up into more-manageable pieces, and by so doing new subroutines have been created. But the actual code has just been moved to a new subroutine and is in most cases essentially unchanged. Subroutine banner, containing code taken from subroutine input, is an example. In several places -- the getdsi... subroutines, the poif... subroutines, the rhsf... subroutines, and in the grid quality package -- the practice of having six different subroutines (one for each face, wherein 1 and 2 are nearly identical, 3 and 4 are nearly identical, etc.) has been done away with. Instead there are now three subroutines (one to do the job for faces 1 and 2, another for faces 3 and 4, etc.). Thus the total subroutine count is smaller that it would otherwise be, and redundant code is removed.

Every subroutine or function called by the program is either contained in the program or to be found in the IGL, with one exception. That exception, which applies only to SGI workstations, is that subroutines dobut82_g.f and zbufit_g.f call the UNIX function "system", which calls the UNIX function "scrsave". If difficulties regarding this arise during linking, the user should simply comment out those calls. In doing so the ability to make screen dumps from within the program will be lost.

PROGRAM FLOW ILLUSTRATED BY A CALL TREE

Following is a call tree for the program. Ideally, the entire call tree would be displayed in one figure, but space doesn't permit. Therefore, detail call trees for certain of the subroutines follow on subsequent pages. The reader can find out all the subroutines or functions a subroutine calls by tracing along all the lines proceeding down and sideways from that subroutine name.

Figure 1. Program Flow Illustrated by a Call Tree.

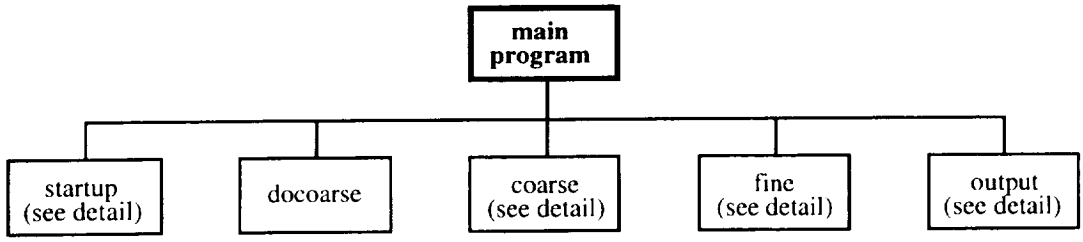

Figure 1a. Subroutines and Function Called by Main Program.

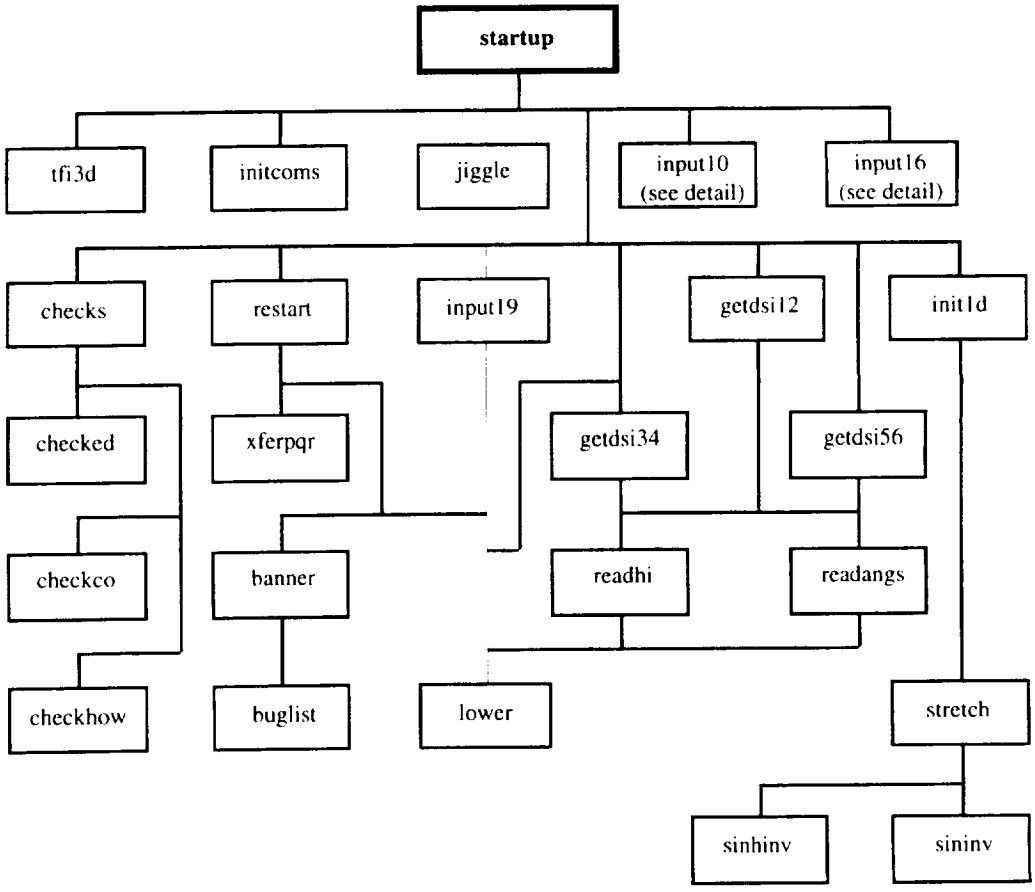

Figure 1b. Detail of Subroutines and Functions Called by Startup.

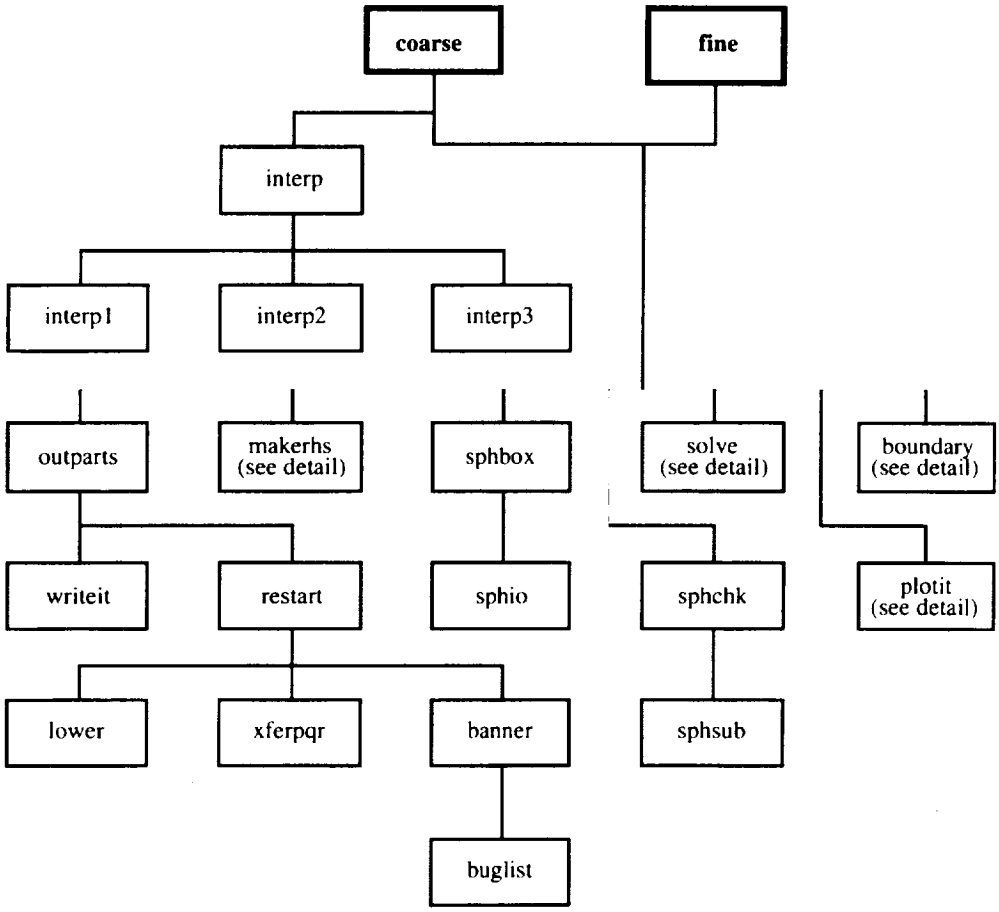

Figure 1c. Detail of Subroutines Called by Coarse and Fine. Plotit Called Only by Graphical Version.

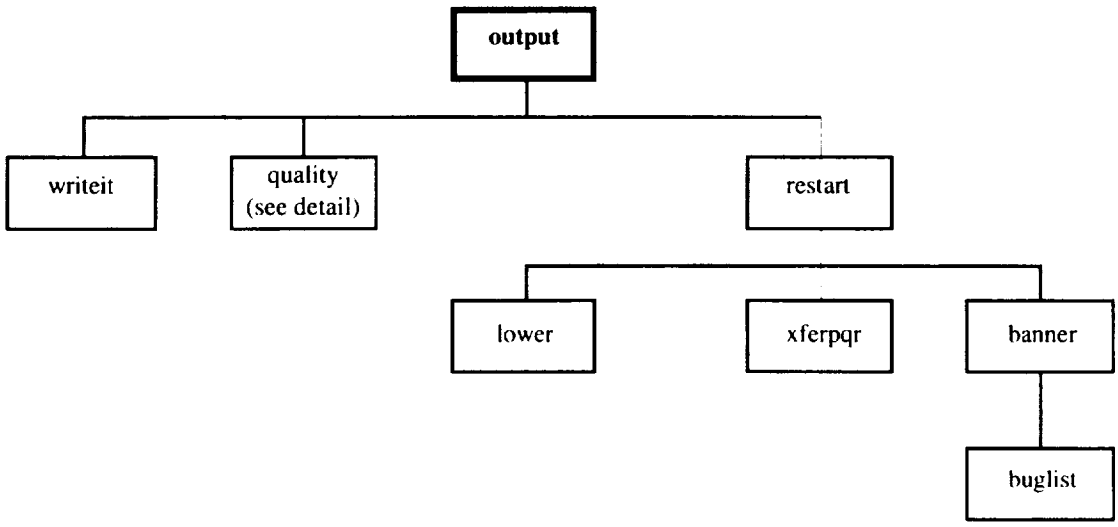

Figure 1d. Detail of Subroutines Called by Output.

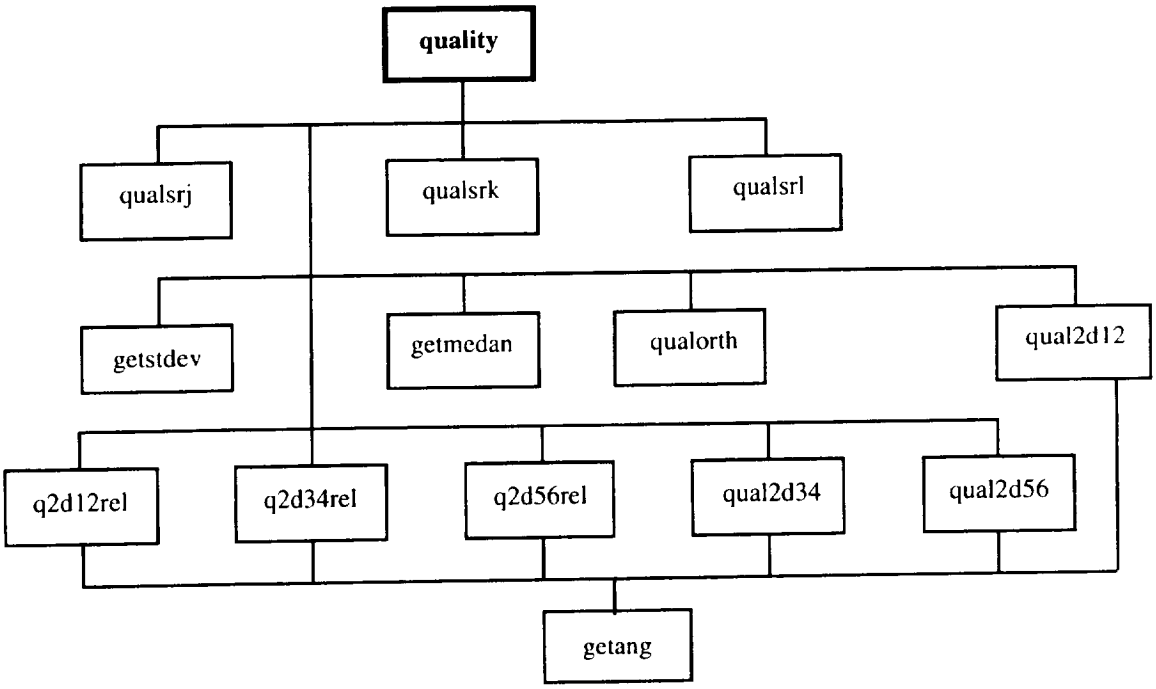

Figure 1e. Detail of Subroutines and Function Called by Quality.

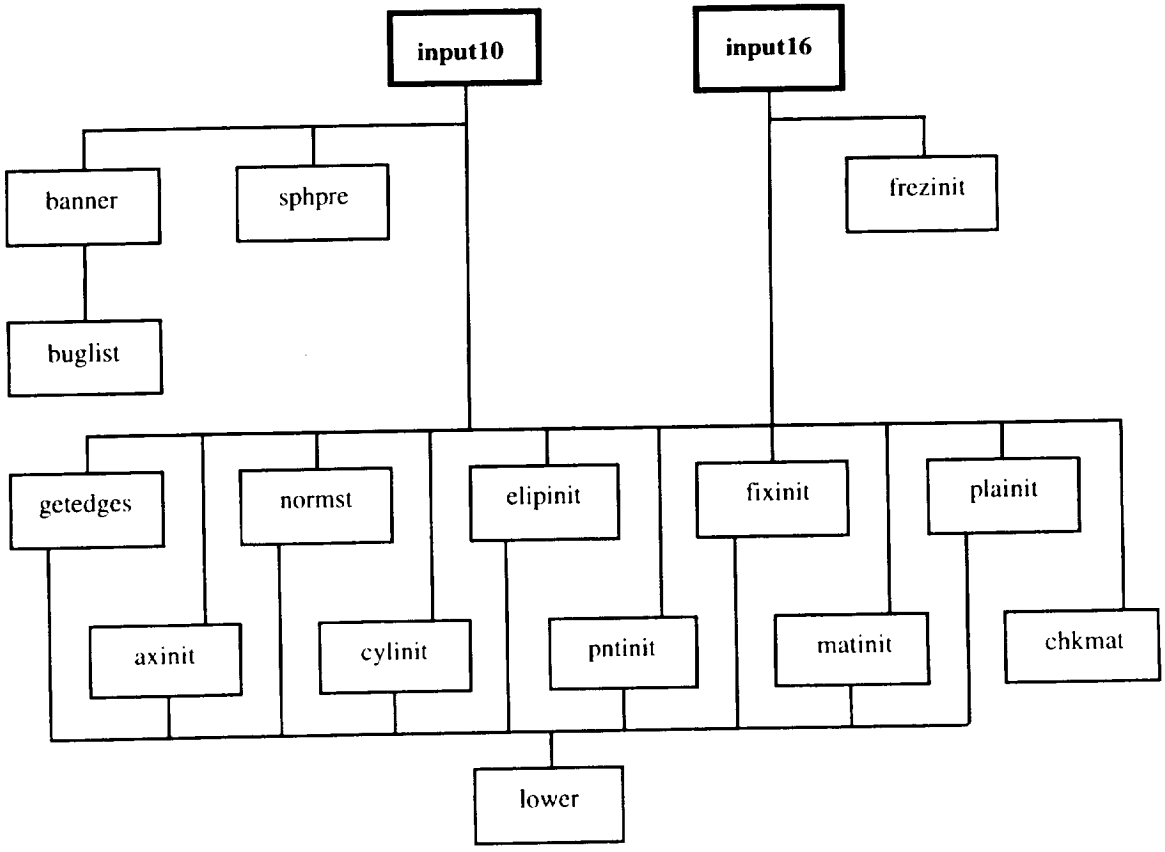

Figure 1f. Detail of Subroutines Called by Input10 and Input16.

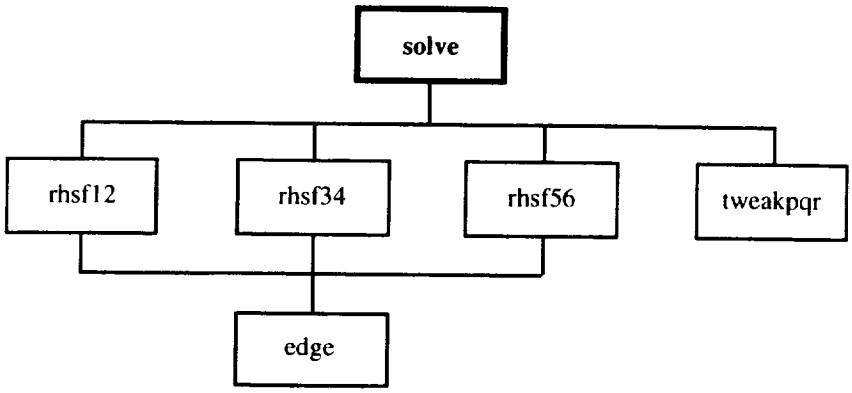

Figure 1g. Detail of Subroutines Called by Solve in Version For Single-Processor Workstations.

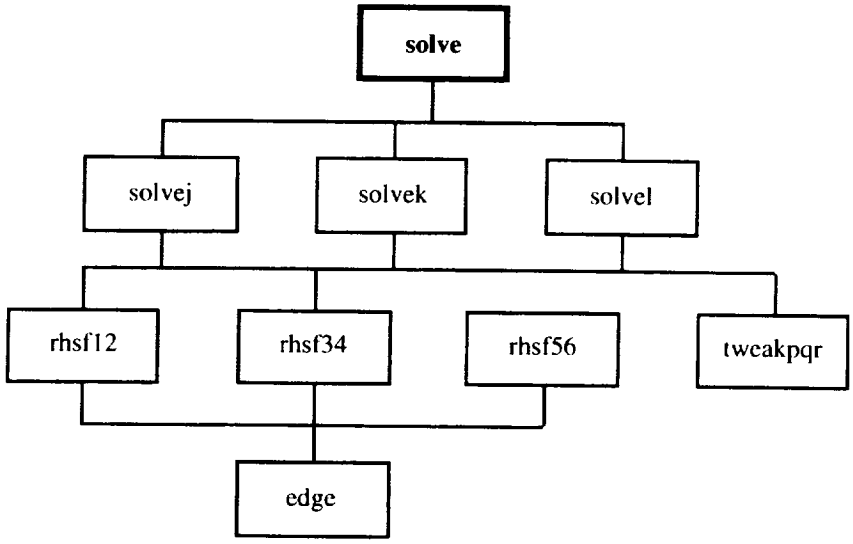

Figure 1h. Detail of Subroutines Called by Solve in Version For CRAYs and Multiple-Processor SGI Workstations.

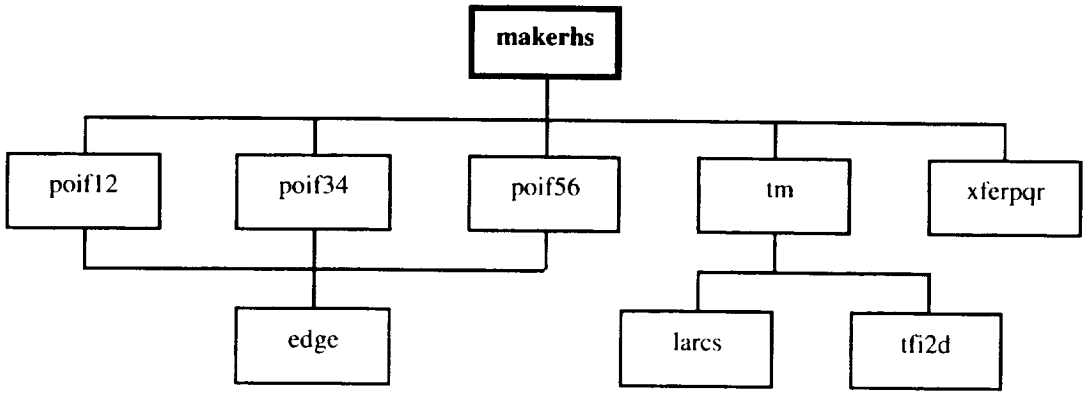

Figure 1i. Detail of Subroutines Called by Makerhs.

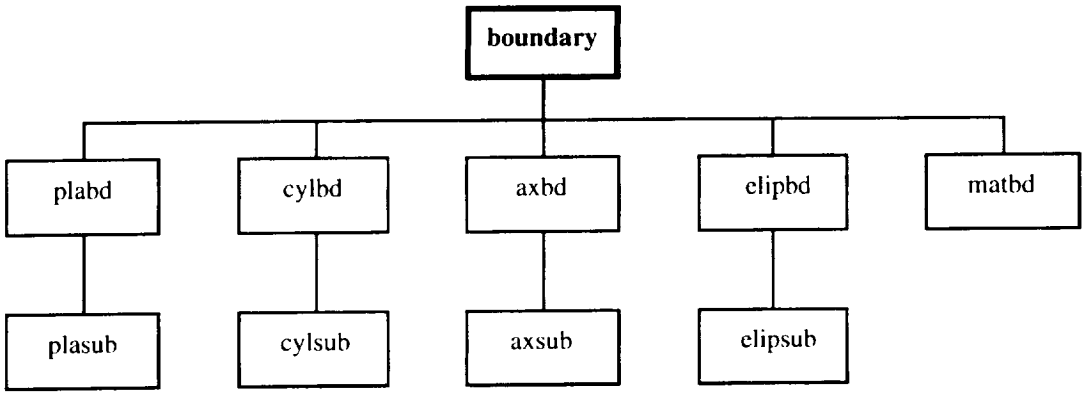

Figure 1j. Detail of Subroutines Called by Boundary.

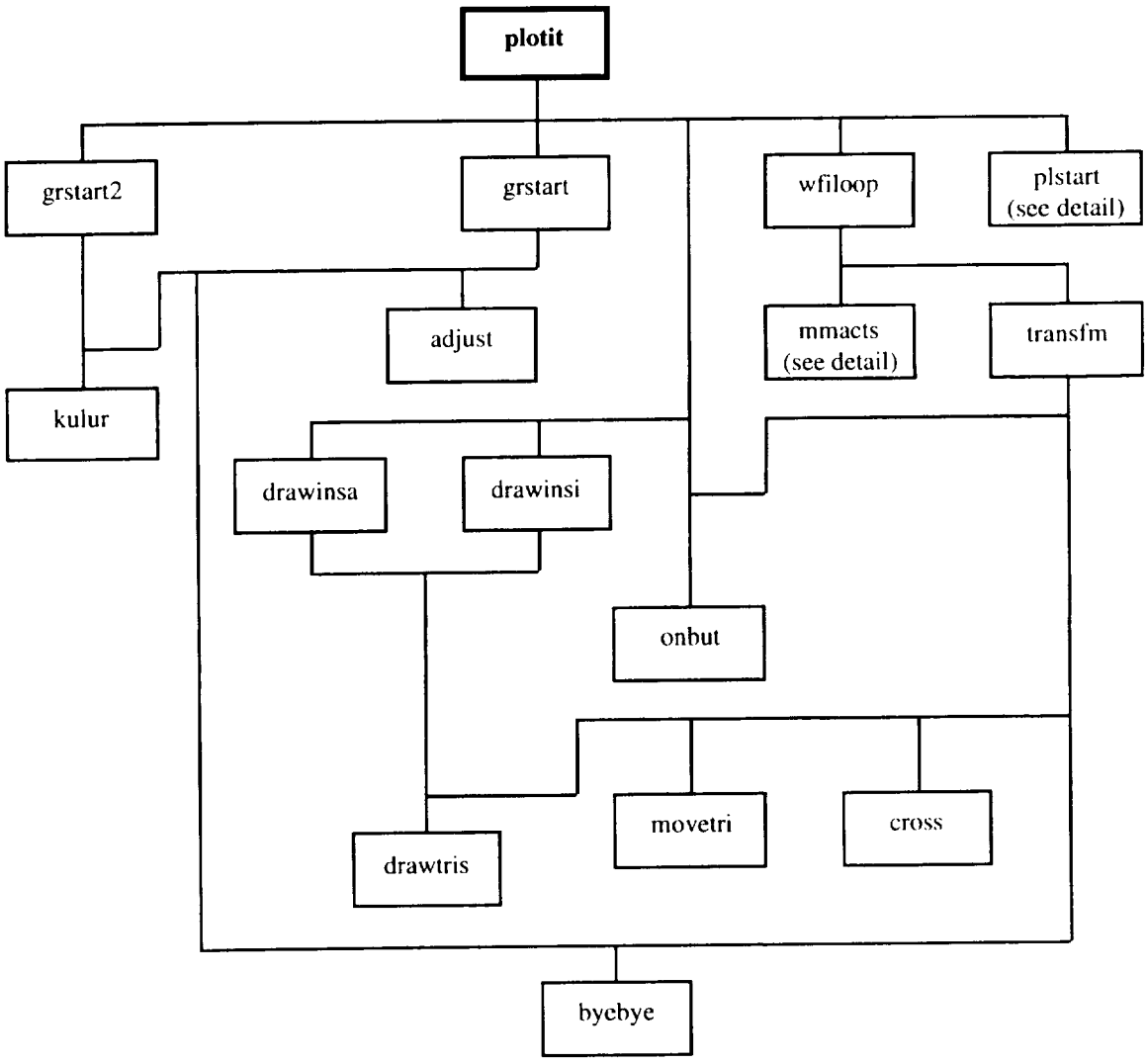

Figure 1k. Detail of Subroutines Called by Plotit. Plotit Called Only in Graphical Version.

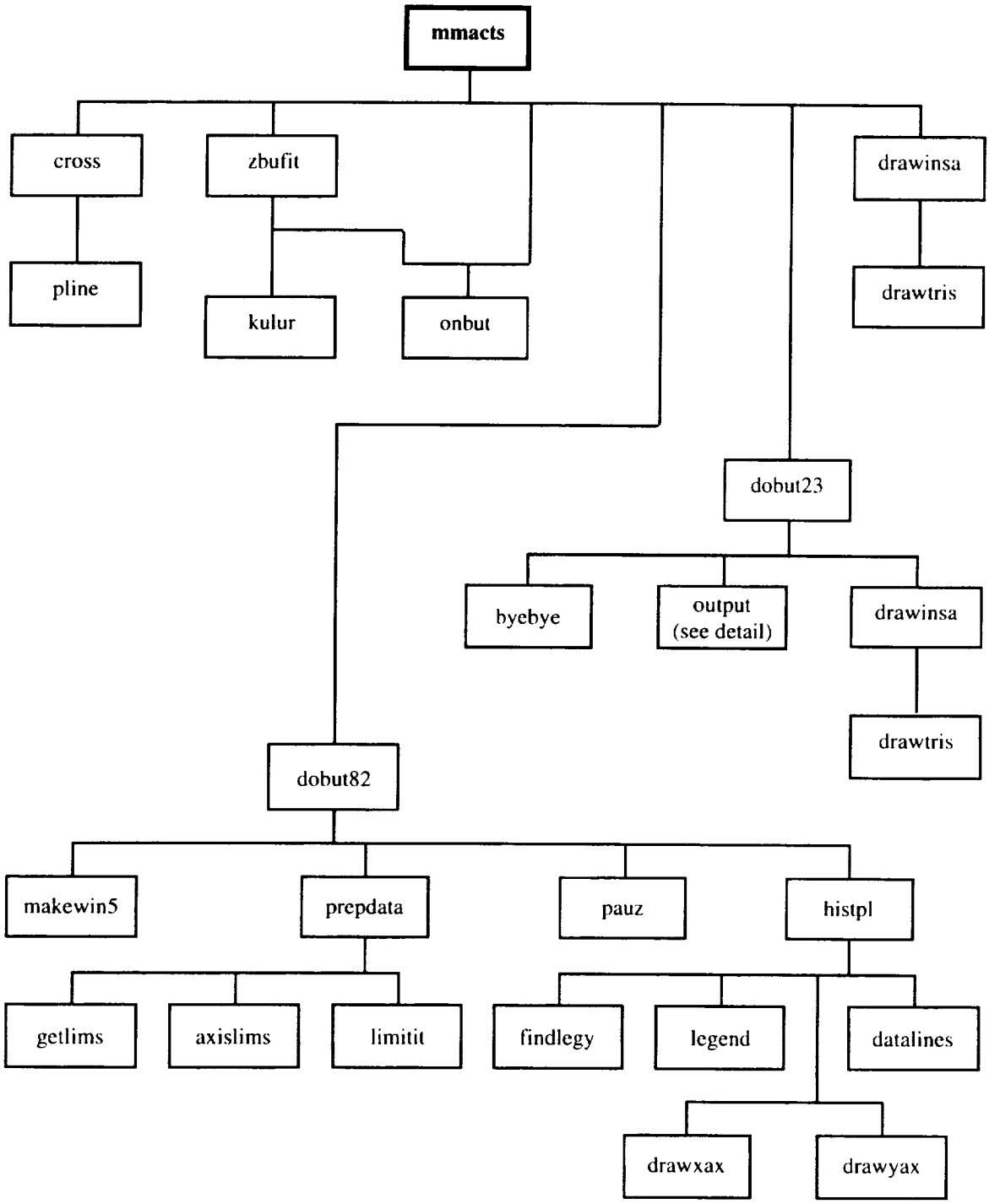

Figure 1l. Detail of Subroutines Called by Mmacts. Mmacts Called Only in Graphical Version.

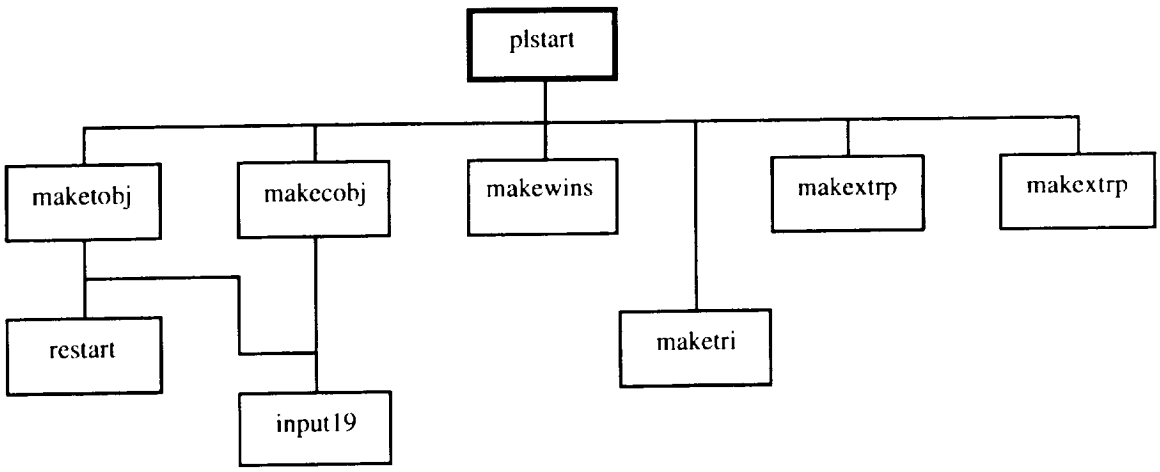

Figure 1m. Detail of Subroutines Called by Plstart. Plstart Called Only in Graphical Version.

The dimension sizes of the arrays, in the common blocks and elsewhere, are specified in the header file "params.h". They are reproduced in the table below. There is nothing special about the sizes as shipped; they can be re-adjusted to any values appropriate to the user's application. As shipped, they will run all the supplied example cases. If they are too big the executable will take up excess space on disk and in memory, and for that reason it might run slower; if they are too small the program will print an error message and quit.

Name of Parameter:	Value, as code is shipped:	Use and meaning:
limpts	200000	Maximum number of points, summed over all blocks
limsrf	25000	Maximum number of controlled points, summed over all faces
limvec	130	Maximum dimension value of j, k and l in any block. Must be less than limsrf.
limblk	5	Maximum number of blocks
limhis	3000	Maximum number of iterations, all parts
limparts	10	Maximum number of parts in the iteration schedule
limviews	50	Maximum number of surfaces per view in graphical version

Table 4. List of Parameters

The program uses ten logical unit numbers for input and output, numbers 10 through 19 inclusive. They are described in the table below:

Logical unit numbers :	Character variables containing the associated file names:	Where those character variables are allocated:	Subroutine in which those unit numbers are used:	Whether they are input or output :	What data is in that I/O:
10	fnamin10	input10.f	input10.f	input	new start control scalars
11	ffnamin11	files.h	fixinit.f	input	X,Y,Z of read-in-fixed surfaces
12	fnamin12	files.h	readhi.f	input	cell heights on controlled surfaces
			readangs.f	input	angles w/r controlled surfaces
13	fnsmooin	input19.f	input19.f	input	read a grid generated elsewhere
14	fnamgrid	files.h	writeit.f	output	write out the finished grid
15	fnamot15	files.h	restart.f	output	write a restart file
16	fnamin16	restart.f	restart.f	input	restart control scalars
17	fnamin17	restart.f	restart.f	input	read a restart file
18	viewsfn	files.h	grstart_g.f	input	indices of surfaces to be viewed
19	fnamin19	input19.f	input19.f	input	smooth start control scalars

Table 5. Logical Unit Numbers Used in this Program

Because Table 5, above, could also be taken as a list of the files read or written by the program it might be appropriate to mention here that the graphics version of the code, documented in a subsequent section, can write screen-dump files for making pictures. Screen dumps of grid pictures will be named plotit.01.rgb, plotit.02.rgb., etc., and screen dumps of convergence history plots will be named convhist.01, convhist.02, etc.

There are comments in the makefiles giving further information about compiling and linking.

Depending on how the program was linked, it is executed by typing either "gral" or "gral_g".

INPUT

THE FIRST TWO LINES

The first thing 3DGRAPE/AL does as it begins execution is to write an interactive prompt asking what kind of a grid generation run this is. There are three acceptable responses. They are read, and then the user is prompted for another datum, the filename from which subsequent data is to be read. A carriage-return causes the program to use its default for this filename. These matters are summarized in the following table:

Interactive response:	Result:	Unit from which subsequent data is read:	Default filename:
"newstart"	A new grid is generated from initial conditions	file10	"file10"
"re-start"	A partly-generated grid is further iterated	file16	"file16"
"smooth"	A grid already generated elsewhere is read in and smoothed	file19	"file19"

Table 6. The First Two Lines of Input Data

The preceding discussion of the first two lines of input assumes that 3DGRAPE is being run on an interactive machine. If it is being run on a batch machine, the prompts will be written to the printout file, along with an echo of the input. The actual input of these two lines in this case will come from the main job input stream. Literally, they are read by the logical unit denoted in the program by an asterisk, as in "read(*,100)...."

When running the program on an interactive machine one can grow weary of typing those first two lines of data. A solution to this is to store those two lines of input data in a file and re-direct it into the program. For example, in running the first sample case, discussed below, the user might put the two lines of input data:

```
newstart
boxall.f10
```

into a file and name it "boxin". The user would then execute the program by typing:

```
gral < boxin
```

or

```
gral_g < boxin
```

FILE10 -- CONTROL SCALARS FOR NEW START

Input on file10 is formatted text, and thus is readable by humans. The records are at most 70 columns wide. All data for file10 must be in exactly the right columns. Column numbers will be clearly delineated below, and they must be followed exactly. There is some consistency here: face numbers will always be read in I1 format, block numbers in I2 format, indices and certain other integers in I3 format, floating-point numbers in F12 format, and file names in A15 format. The reading of static text and character strings is case-insensitive, meaning that it ignores whether letters are in upper- or lower-case. There is one exception to this: filenames, read on a UNIX system, are case-sensitive. When a "line" of input data must be continued, the continuation line always begins with three dots. There are three places in the file10 input where blank lines are allowed: before and after "iterations" lines, before "block" lines, and before "face" lines. Blank lines in these places can greatly enhance clarity and readability.

There are places in the input where the user is given the option of entering either a character string or a floating-point number. The program is smart enough to sort out that form of input. It was stated earlier that floating-point numbers are read in F12 fields. To be precise, the format specification is F12.0. But that does not mean that only whole numbers may be read. According to the rules of Fortran, a decimal point in an input record overrides any placement of the decimal point implied by the format statement. Thus the user may put a decimal point anywhere in the floating-point input fields.

The discussion of each input will be preceded by a list of all relevant data. Note that some input lines require continuation. The first column in this list is the line number. The next column gives the range of column numbers for each field. Then a code will indicate what type of datum this is:

- "st" for static text (a character string which should be entered exactly as stated, and which is required for readability),
- "i" for integer,
- "f" for floating-point number,
- "n" for file name
- "c" for a character string

In some places the user may put into a field either a character string or a number. Those codes are:

- "c/f" a character string or a floating point number
- "c/i" a character string or an integer

The fourth column contains a brief description of what that datum is. The table will be followed by one or more examples above a column number key. After that will follow a discussion of the indicated input line(s).

The input 3DGRAPE/AL expects to read from file10 begins with several lines which give information about the entire grid and about the entire run of 3DGRAPE/AL. It then goes into an outer loop on block number, and for each pass it reads information about the block. Inside that is an intermediate loop on face number and for each pass it reads information about the face. Inside that is an inner loop on section number within the face, reading information about each section. At the conclusion of those nested loops, it is finished reading from file10.

Because the program will know the number of blocks, and knows that there are six faces per block, and will know into how many sections the faces are divided, it will know when

all the required input data have been read. I.E., it will know when to stop reading. Therefore the user can store other input data records, not currently in use, below the last line to be actually read. Users have found this convenient.

The "run-comment" lines

Line no.:	Column nos:	Datum type:	Description:
	1-20	st	"run-comment "
	21-70	c	free-field comment describing this run

```
run-comment          Blah, blah, blah.
run-comment          What this data is all about.
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

The file10 input begins with exactly two of these lines. The comments on them will annotate the printout file, and they will help the user to remember what each file10 dataset was used for.

The "number-of-blocks" line

Line no.:	Column nos:	Datum type:	Description:
	1-17	st	"number-of-blocks="
	18-19	i	number of blocks in this grid
	20-28	st	"-heading="
	29-31	c/i	printout heading repetition code
	32-50	st	"-filename-18-views="
	51-65	n	name of file for input as file18

```
number-of-blocks=01
number-of-blocks=01-heading=kdf
number-of-blocks=01-heading=000
number-of-blocks=01-heading=054
number-of-blocks=01-heading=kdf-filename-18-views=my_picture_data
12345678901234567890123456789012345678901234567890123456789012345
         11111111112222222222333333333344444444445555555555666666
```

The printout features a convergence history for each block. There is a heading telling what data are in the columns of numbers. By the default, obtained by putting blank, zero, or kdf (meaning "keep default") in cols. 29-31, the heading is printed once per block. If

the user wants this heading to be printed more often, the number of lines of data between headings should appear in this field.

From file18 the program reads input describing just what points are to be drawn in each of the views of the iterating grid as shown by the graphical version of the code. Non-graphical versions of the code ignore this field. The contents of this file are described in a subsequent section.

Everything on this line after column 19 is optional.

The "iterations" lines

The "iteration lines" are actually a set of three lines, with there being two options for the form of the second line. This set of three lines is repeated for each "part" in the iteration schedule (this is explained below). The first line:

Line no.:	Column nos:	Datum type:	Description:
1	1-11	st	"iterations="
1	12-14	i	the number of iterations in this part
1	15-23	st	"-control="
1	24-25	c	overriding global switch on control, either "ye" or "no"
1	26-35	st	"-rhs-type=
1	36-47	c	RHS type (see below)
1	48-60	st	"-coarse/fine="
1	61-66	c	"coarse" or "fine "

```
iterations=100-control=no-rhs-type=S&S-initzero-coarse/fine=coarse
iterations=100-control=no-rhs-type=keep-default-coarse/fine=fine
iterations=100-control=ye-rhs-type=S&S-init-T&M-coarse/fine=fine
iterations=100-control=ye-rhs-type=Thomas&Middl-coarse/fine=fine
iterations=100-control=ye-rhs-type=blendS&S&T&M-coarse/fine=fine
iterations=100-control=ye-rhs-type=s&s-continue-coarse/fine=fine
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
         11111111112222222222333333333344444444445555555555666666666
```

Option 1 for the second line:

Line no.:	Column nos:	Datum type:	Description:
2	1-20	st	"...relax-param-type="

2	21-25	c	For this option, the character string "fixed"
2	26-44	st	"-relax-param-value="
2	45-56	c/f	value for uniform relaxation parameter

```
...relax-param-type=fixed-relax-param-value=keep-default
...relax-param-type=fixed-relax-param-value=  1.0
12345678901234567890123456789012345678901234567890123456
         11111111112222222222333333333344444444445555555
```

Option 2 for the second line:

Line no.:	Column nos:	Datum type:	Description:
2	1-20	st	"...relax-param-type="
2	21-25	c	For this option, the character string "optim"
2	26-39	st	"-recomp-every="
2	40-42	c/i	recomputation interval
2	43-52	st	"-how-much="
2	53-64	c/f	scale factor for optimum relaxation parameter

```
...relax-param-type=optim-recomp-every=kdf-how-much=keep-default
...relax-param-type=optim-recomp-every=010-how-much=   0.75
123456789012345678901234567890123456789012345678901234567890123456789012345678901234
         11111111112222222222333333333344444444445555555555566666
```

The third line:

Line no.:	Column nos:	Datum type:	Description:
3	1-16	st	"...abc-override="
3	17-28	c/f	overriding abc value

```
...abc-override=no
...abc-override=123456789.12
123456789012345678901234567890
         111111111222222222
```

This program has the ability to divide the iterations it will do in a run into parts, with all the data shown on these lines being variable between parts. That can enable the experienced user to accomplish as much in one execution of this code than in several executions of other codes. A set of these three lines should be present for each part in the

iteration schedule. The maximum number of parts allowed is dependent upon a parameter set in params.h; that parameter is set to 10 as the code is delivered.

The "face" lines, described in a subsequent section, turn the control of cell height and skewness on or off for each face, and specify what cell height is being requested if the control is on. The "control" parameter on the first line of this set of "iteration" lines is a global switch which overrides whatever is found on the "face" lines. If this switch is on, then the control is on or off according the individual "face" lines; if this switch is off then there will be no control regardless of that the "face" lines say. With no control, i.e., with the RHS terms remaining at zero, the Poisson Equations become the Laplace Equations. These give rise to a grid which is smooth, but has cell heights tending to be uniform and has no tendency toward orthogonality.

A tip --> It is highly recommended that the user start with one part, having a few iterations (e.g., 5) and no control. This should either be done with the graphical version, or the grid should be iterated in batch and then examined using other graphical tools. The boundary points and several representative interior surfaces of each block should be observed. After zero iterations (i.e., as of the initial conditions) the grid may look rather strange, but that strangeness should go away during the first iteration or two. After that the user can verify that the boundary data and boundary treatments are correct. These boundary conditions are the source of many of the input errors users make, and it is much easier to find and correct them at this point rather than after the complications of RHS terms and long iteration runs have been added.

The user might then want to iterate an uncontrolled solution to convergence, which should further exonerate the boundary data and conditions. After this is seen to work, the user should re-generate the grid, with control terms.

There are choices for which type of Right-Hand-Side terms (also known as "control terms") is to be used in each iteration part. See columns 36-47 in the first line of this set. They are:

- "keep-default" -- This, as it says, causes the program to use the default RHS type. For the first iteration part of a new start, this is the same as "S&S-initzero" (see below). For a subsequent iteration part, or at the start of a restart run, it is the same as "S&S-continue" (see below).
- "S&S-initzero" -- Thus causes the Steger & Sorenson (S&S-type) RHS terms to be used. They are initialized to zero, and then updated iteratively at teach time step. They converge to the values which give the desired cell height and skewness at boundary surfaces, with control effects decaying exponentially with distance from the boundary surface.
- "S&S-init-T&M" -- This is just like "S&S-initzero" except that the terms are initialized to the values computed by the Thomas & Middlecoff (T&M-type) method. This can speed convergence by providing better initialization of the RHS terms. But note that T&M-type requires that all six faces of the block consist of fixed points in space (rather than floating around on some surface), hence this choice for initialization of the S&S-type terms has the same limitation.
- "S&S-continue" -- This option assumes that it is a subsequent iteration part or a restart run, and that S&S-type terms were used previously. It simply continues to iterate on the S&S-type terms, further refining them.

- "Thomas&Middl" -- Use the Thomas & Middlecoff RHS terms. All six faces of the block must consist of fixed points in space (rather than floating around on some surface).
- "blendS&S&T&M" -- This option uses both S&S-type and T&M-type RHS terms, blended as a function of the distance from the boundary surfaces. It seems that S&S-type terms give the best results near the boundary surfaces, and that T&M-type gives the best results in the interior of the blocks, and this blending attempts to use that principle to give the best of both. Note that the limitation that all six faces be fixed points in space applies here, as with all uses of the T&M-type terms.

If the control terms are turned off, then this datum telling which type of control terms are to be used is ignored.

This code includes an artifice to accelerate convergence, which is called "coarse-fine grid sequencing." It is like one-half of a pass through a multigrid solver.[10] Every third point in each direction (which together make up the "coarse grid") is used in a course iteration step; all points (which together make up the "fine grid") are used in a fine step. To employ this technique the user should specify coarse steps and iterate to convergence, including the use of RHS terms as desired. The user should follow that with iteration to convergence again, this time using fine steps. The program will automatically interpolate from the coarse data to make an initial fine grid after the last coarse part. Any number of coarse steps may be followed by any number of fine steps, but coarse solution steps may not follow fine steps.

The logic here is that the coarse solution will go fast because it does approximately one-twenty-seventh as much arithmetic per step, and that the fine solution should go fast because it starts with initial conditions which differ from the final solution only by errors introduced during the interpolation between coarse and fine. The effectiveness of this technique varies greatly from case to case, but the user can count on a reduction in CPU time of at least 50%, sometimes much more. There is a drawback, and it is that if coarse-fine grid sequencing is to be used the maximum number of points in each of the three coordinate directions in every block must be of the form 3n+1 for n some integer greater than or equal to 4 (example: 13, 16, 19, 22, etc.). In some cases this requirement is found to be burdensome, and so the use of this speedup procedure is not possible.

The Poisson equation solver in 3DGRAPE/AL uses a point-Successive-Over-Relaxation (point-SOR) algorithm. This algorithm has in it a "relaxation parameter" which controls the speed at which the solution process is driven. This parameter, usually given the name Omega (Ω), varies between 0. and 2. If Ω is set too low, the solution will take an excessive number of iterations; if it is set too high the solution process will "blow up" and no solution will be found. The simplest way to set Ω is to use a fixed value. Experience has shown 0.7 to be a safe choice, and that is the default. The user may select this, or any other fixed value, using option 1 for the second line in this set.

---

[10] An attempt was made to implement a true multi-grid solver in this code, but it didn't work. It was determined that the basic reason for the failure was that in the finished grid the cell heights in the direction normal to the boundary surface increase in a generally exponential fashion with distance from the surface, but the exact rate of that increase is indeterminate. Thus, although the user specifies the spacing between the boundary and the first node in the field, it is not possible to specify the spacing between the boundary and the 2^{n-1}-st node in the field for n>1. That spacing is needed to formulate the RHS term at the n-th multigrid level. An estimate can be made, but it is not accurate. Thus, there was a fundamental inconsistency between the RHS terms being solved at the different multigrid levels. The different multigrid levels were attempting to solve what were, in effect, different equations. Obviously, such an algorithm would not converge.

Option 2 for the second line in this set uses a locally-varying and time-varying optimum relaxation parameter. Use of an optimum Ω can minimize the number of iterations required to find the grid solution. It is calculated using the *Ad-Hoc* method of Erlich. (Ref. 8). This method requires a significant amount of calculation to find the Ω at every time step, and that can use up time saved by Ω being optimum. One solution to this conundrum is to re-compute the optimum Ω at intervals. The "recomputation interval" is the number of steps in that interval; it's default value is 10. Lastly, there is the scale factor for the optimum relaxation parameter. Experience has shown that Ω calculated this way can sometimes be a little too big, causing blow up. Therefore the user is given a scale factor by which the Ω is multiplied; it's default is 0.75.

Just as the "face" lines (described below) give the desired cell heights and the "control" parameter on this first of these three lines gives a global override for it, similarly, the face lines give values for the abc parameter (also described below) and the third line here gives a global override for it. If a number appears on this line, it will take precedence over the abc parameter values specified on the face line. The word "no" in that field causes the abc parameter values specified on the face lines to be used.

The "filename-11" line

Line no.:	Column nos:	Datum type:	Description:
	1-18	st	"filename-11-input="
	19-33	n	name of file for input as file11
	34-52	st	"-filename-12-output="
	53-67	n	name of file for input as file12

```
filename-11-input=my_xyz_data    -filename-12-input=my_cell_heights
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
         11111111112222222222333333333344444444445555555556666666666
```

The data in file11 are the X,Y,Z coordinates of points on boundary surfaces, which are supplied from another source. They are described in detail in a subsequent section.

Users familiar with the input for the earlier 3DGRAPE code should note that logical unit 12, designed for debugging in the earlier code and rarely used, has a completely new meaning here. In 3DGRAPE/AL unit 12 may, depending on the file10 input, be used to input the cell heights and skewness of grid cells on boundaries. Its format is described in a subsequent section.

The "filename-14" line

Line no.:	Column nos:	Datum type:	Description:
	1-24	st	"filename-14-grid-output="

25-39	n	filename for main grid output
40-45	st	"-form="
46-52	c	"3dgrape" or "plot3ds" or "plot3dm" or "charact"

```
filename-14-grid-output=my_grid_file    -form=3dgrape
filename-14-grid-output=my_grid_file    -form=plot3ds
filename-14-grid-output=my_grid_file    -form=plot3dm
filename-14-grid-output=my_grid_file    -form=charact
12345678901234567890123456789012345678901234567890123456789012
         1111111111222222222233333333334444444444555
```

The main grid output may take any one of four different forms. The first is a form designed for this program, called "3dgrape". It is best described by the following pseudo-code:

```
      open(unit=14,status='new',form='unformatted',file='my_grid_file')

      write(14) maxblk

      do nblk=1,maxblk

       jmax=jmaxa(nblk)
       kmax=kmaxa(nblk)
       lmax=lmaxa(nblk)

       write(14) jmax,kmax,lmax

       write(14) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     1           (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2           (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)

      enddo

      close(unit=14)
```

If "plot3ds" is specified the data on file14 are written in the form required by the well-known NASA graphics program PLOT3D, using its single-block option. If "plot3dm" is specified the data on file14 are written in the PLOT3D format, using its multiple-block option. These options, also, are best seen in pseudo-code:

```
      open(unit=14,status='new',form='unformatted',file='my_grid_file')

      if(maxblk.gt.1)  write(14) maxblk

      write(14) (jmaxa(nblk),kmaxa(nblk),lmaxa(nblk),nblk=1,maxblk)

      do nblk=1,maxblk

       jmax=jmaxa(nblk)
       kmax=kmaxa(nblk)
       lmax=lmaxa(nblk)

       write(14) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
```

```
     1            (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2            (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)

      enddo

      close(unit=14)
```

If "charact" is specified, the data on file14 are written as formatted data. This is useful for users running on computers connected to a network which does not have the facility to transfer binary data. A main grid output file created this way will be several times as large as if either of the three other options had been used, and it will take several times as long to read and write, but for some users this approach is unavoidable. This form is essentially the "3dgrape" form converted to formatted output:

```
      open(unit=14,status='new',form='formatted',file='my_grid_file')

      write(14,100) maxblk

      do nblk=1,maxblk

       jmax=jmaxa(nblk)
       kmax=kmaxa(nblk)
       lmax=lmaxa(nblk)

       write(14,100) jmax,kmax,lmax
100    format(3i10)

       write(14,101) (((x(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     1               (((y(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax),
     2               (((z(j,k,l,nblk),j=1,jmax),k=1,kmax),l=1,lmax)
101    format(5e15.6)

      enddo

      close(unit=14)
```

The "write-for-restart" line

Line no.:	Column nos:	Datum type:	Description:
	1-18	st	"write-for-restart="
	19-20	c	either "ye" or "no"
	21-40	st	"-filename-15-output="
	41-55	n	filename for restart file

```
write-for-restart=no-filename-15-output=my_restart_file
123456789012345678901234567890123456789012345678901234
        11111111112222222222333333333344444444444555555
```

3DGRAPE/AL has a restart capability. This should not be confused with the parts in the iteration schedule. Parts in the iteration schedule are completed during one run; the restart capability allows it to make more than one run. The user can run it a while, and then decide to run it some more, either with or without some changes. More things can be changed at a restart than when going between parts; everything appearing in files 11 and 16, described in subsequent sections, can be changed at a restart.

To make restart possible, the code writes a file containing all it needs to continue where it left off. File15 is that file. It is output by the code in the run before the restart, and then read back in on the restart run. It is a very large file, containing the contents of most of the common arrays, and some other material as well. The file is unformatted, and so not readable by humans.

The character-string "ye" or "no" in columns 19-20 determines whether the file is to be written. The file name appears in columns 41-55.

The "omegpqr" line

Line no.:	Column nos:	Datum type:	Description:
	1-8	st	"omegpqr="
	9-20	c/f	relaxation parameter for S&S-type RHS terms
	21-28	st	"-pqrlim="
	29-40	c/f	growth limit factor for S&S-type RHS terms

```
omegpqr=keep-default-pqrlim=keep-default
omegpqr=123456789.12-pqrlim=123456789.12
123456789012345678901234567890123456789012345678
         11111111112222222222333333333344444444
```

If the user chooses S&S-type RHS terms, the code iterates to find them at the same time that it iterates to find the X,Y,Z. There is an Ω for the S&S-type RHS terms, just as for the X,Y,Z, although its size range is different. The default value is 0.3. There is another input parameter affecting iterations for finding the S&S-type RHS, and that is "pqrlim". The S&S-type RHS terms tend to blow up in the first few iterations, so their growth rate is limited. Their absolute value may grow by not more than this parameter times their value at the previous time step.[11] The default value for this is 0.5.

The "quality-check" line

Line no.:	Column nos:	Datum type:	Description:
	1-14	st	"quality-check="

[11] This has the curious effect of causing their growth to be limited by an upward slanting straight line on a semi-log plot, when plotted as a function of iteration count.

15-16	c	"ye" or "no"
17-30	st	"-output-after="
31-35	c	"parts" or "done "

```
quality-check=ye
quality-check=ye-output-after=parts
quality-check=no-output-after=done
12345678901234567890123456789012345
         11111111112222222222333333
```

The program includes a grid quality evaluation feature which computes and prints maxima, minima, medians, and averages of cell heights and non-orthogonality, at boundaries and in the interiors of the blocks of the finished grid. It requires some CPU time (approximately the same as one-and-one-half fine iterations), and it generates several pages of output. The datum in columns 15-16 turns this feature on and off.

By default, the program writes the grid solution file (file14) and the restart file (file15) once, after finishing all the parts in the iteration schedule. But users using large amounts of computer time may wish to save their work after each part in the iteration schedule. If "parts" is found in cols. 31-35, the grid solution file, along with the restart file if a restart file is called for, will be written after each part in the iteration schedule. The file names used will be those given above, with appending characters giving the number of the iteration part after which they were written. For example, if "parts" is selected here, and "my_grid" is given for the name of file14, and three parts are used in the iteration schedule, the resulting grid solution files will be "my_grid.1", written after the first iteration part, "my_grid.2", written after the second iteration part, and "my_grid", written after the last iteration part. "Done" or blanks in cols. 31-35 cause the files to be written only once, after the last iteration part.

Everything after column 16 is optional.

The foregoing input data records give information about the entire grid-generation operation being conducted by this run of 3DGRAPE/AL. Following these lines the program goes into an outer loop on the block numbers. For each block a group of lines must be read which give characteristics of the block.

The "block-comment" line

Line no.:	Column nos:	Datum type:	Description:
	1-6	st	"block-"
	7-8	i	number of this block
	9-20	st	"-comment "
	21-70	c	free-field comment describing this block

```
block-01-comment   Blah, blah, blah.
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

The comment in the comment field of the block statement will be used to annotate the printout. The printout will include a convergence history for each block, labeled with these comments.

The "dimension" line

Line no.:	Column nos:	Datum type:	Description:
	1-12	st	"dimension-j="
	13-15	i	maximum value of first subscript, j
	16-28	st	"-dimension-k="
	29-31	i	maximum value of second subscript, k
	32-44	st	"-dimension-l="
	45-47	i	maximum value of third subscript, l

```
dimension-j=019-dimension-k=031-dimension-l=022
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

The dimensions of each block are variable, and may be set by the user at execution time. The dimension sizes must in every case be at least 4. If "coarse" iteration steps are to be performed, then the dimension sizes must be of the form 3n+1 for n some integer greater than or equal to 4. The upper bound on these dimension sizes is indirect. They determine the total number of points in the grid, which is limited by one of the parameters in "params.h", which is limited by the memory of the computer on which the program is installed.

The "handedness" line

Line no.:	Column nos:	Datum type:	Description:
	1-11	st	"handedness="
	12	c	either "r" or "l"
	13-22	st	"-initcond="
	23	c	either "j" or "k" or "l" or "t" (see below)

24-33	st	"-cart/sph="
34-42	c	either "Cartesian" or "spherical"
43-50	st	"-numtfi="
51-53	c/i	number of TFI iterations (see below)

```
handedness=r-initcond=j-cart/sph=cartesian
handedness=r-initcond=k-cart/sph=spherical
handedness=r-initcond=l-cart/sph=cartesian-numtfi=kdf
handedness=l-initcond=t-cart/sph=cartesian-numtfi=010
12345678901234567890123456789012345678901234567890123
         11111111112222222222333333333344444444445555
```

The "handedness" of the grid -- either right-handed or left-handed -- may vary from block to block. For Laplacian grids it is irrelevant. But for grids with control activated it is used to choose the sign of a square root in the computation of the S&S-type RHS terms, so it must be set properly.

The handedness of a grid can be determined according to the right-hand rule, or in the following equivalent way. Choose any point (j,k,l). A unit vector in the ξ direction is a vector from that point to the point (j+1,k,l). Similarly, a unit vector in the η direction is a vector from (j,k,l) to (j,k+1,l), and a unit vector in the ζ direction is from (j,k,l) to (j,k,l+1).

The three vectors will be bound tail-to-tail-to-tail at the point (j,k,l). Imagine them defining the axes of a locally Cartesian ξ,η,ζ coordinate system. Imagine an ordinary screw, placed coincident with the ζ axis. Then imagine rotating some point on the head of that screw from the positive ξ axis to the positive η axis. If that rotation produces movement of the screw in the positive ζ direction, then the grid is right-handed. If that rotation produces movement in the negative ζ direction, then the grid is left-handed.

When using spherical topology (see below), X,Y,Z coordinates of each point in the block are converted to spherical coordinates ρ,θ,ϕ. This transformation can sometimes cause the handedness to be reversed, in which case the handedness on this input line must be reversed. The symptom of this problem is that when the RHS terms are activated utterly nonsensical cell heights at the body rapidly emerge, either much too large or having negative volumes. If this happens, as it did in the hemisphere-cylinder-cone example case which is shown in a subsequent section, the user should simply reverse the handedness in the input data for the block.

The character in column 12 should indicate that handedness: "r" for right-handed or "l" for left-handed. Users frequently make mistakes on this point, with the result being grids with lines repelled from the controlled faces rather than attracted. Rather than agonize analytically over this point, the user encountering such symptoms might want to simply reverse the handedness and try again.

In starting an execution of the grid generator, once points have been initialized in some way on all six faces of the block, the need arises to initialize the points inside the block. There are two options here. The first is to have the interior points distributed between opposing boundary points on a straight line, with spacing along that line determined by

Vinokur's two-ended stretching algorithm.[12] The user chooses the coordinate direction in which that distribution is to be applied by entering j, k, or l in column 23.

The second option for distributing the points in the interior of the blocks is to use Three-Dimensional Trans-Finite-Interpolation (TFI). To use this option, the user should place "t" in column 23. Note that the use of TFI in this code requires that all six faces of the block consist of fixed points. Therefore, if any block has a face which has points floating about on an analytical surface, or a face which is a block-to-block boundary face, this option cannot be used.

It has been stated that 3DGRAPE/AL should be able to make a grid in any region into which a cube or cubes can be warped. This is true, but for cases having spherical topology, i.e., having a spherical axis , certain mathematical singularities occur and special measures must be taken. The coordinates in such zones are transformed from Cartesian coordinates X,Y,Z into spherical coordinates ρ,θ,ϕ. An iteration is performed on the grid in that space. Then the outermost four shells (or cubic surfaces) are converted back to Cartesian coordinates. Boundary conditions are applied, and the surfaces are transformed back into spherical coordinates. This is iterated to convergence, and the entire block is transformed back into Cartesian coordinates before being written out.

To utilize this option in any block, the user should put "spherical" into columns 34-42. Otherwise, "cartesian" should be entered in those columns. The spherical axis must be coincident with one of the coordinate axes.

The 3-D TFI algorithm iterates to optimize the volume distribution, and the datum in cols. 51-53 is the number of iterations used. "kdf" here, meaning "keep default," causes 10 iterations to be used. This datum is only referenced if "t" is placed in column 23; otherwise it is ignored.

The "polar-axis" line

Line no.:	Column nos:	Datum type:	Description:
	1-11	st	"polar-axis="
	12	c	either "x" or "y" or "z"
	13-19	st	"-along="
	20	c	either "j" or "k" or "l"
	21-28	st	"-around="
	29	c	either "j" or "k" or "l"
	30-37	st	"-center="
	38-49	f	location on polar axis of spherical center

[12] Vinokur, M., "On One-Dimensional Stretching Functions for Finite-Difference Calculations," J. Comp. Phys., vol. 50, no. 2, May 1983, pp. 215-234.

```
polar-axis=x-along=k-around=l-center=   100.
12345678901234567890123456789012345678901234567890123456789
         1111111111222222222233333333334444444444
```

This line is read only if "spherical" appears on the preceding line. In that case, 3DGRAPE/AL needs to know which axis is the polar axis. That datum is entered in column 12. The program then needs to know which index runs along that axis, entered in column 20, and which index runs around it, entered in column 29. In the spherical case neither the body nor the outer boundary need be exactly spherical, but they should be somewhat similar to a sphere. Given that, it should be possible to locate an approximate center to that sphere. That center would, of course, lie on the spherical axis. The location of the approximate center is given by entering its location on the axis in columns 38-49.

The "freezeblock" line

Line no.:	Column nos:	Datum type:	Description:
	1-12	st	freezeblock=
	13-15	c	"yes" or "no "

```
freezeblock=no
freezeblock=yes
12345678901234567890123456789012345678901234567890123456789
         1111111111222222222233333333334444444444
```

In generating a large, multiple-block grid, it is sometimes advantageous to be able to freeze some of the blocks while continuing to iterate on others. If "no " or blanks are placed in columns 13-15, the block will be iterated, as is normally the case. If "yes' is placed in those columns, this block will not be iterated, and will be frozen. This line must be present, or an error will result.

This concludes the inputs which give characteristics of the block. At this point 3DGRAPE/AL goes into an intermediate loop on the six faces of the computational cube. It expects to read information which applies to each face. Blank lines may appear before a "face" line.

The "face" line

Line no.:	Column nos:	Datum type:	Description:
	1-5	st	"face-"
	6	i	face number
	7-13	st	"-sects="
	14-15	i	number of sections into which this face is divided

16-23	st	"-normal="
24-35	c/f	"uncontrolled" or cell height or "n-i-stations" or "read-each-pt" (see below)
36-40	st	"-abc="
41-52	c/f	"keep-default" or stretching parameter
53-58	st	"-angs="
59-70	c	"keep-default" or "default+edge" or "def-1-read-2" or "read-1-def-2" "read-each-pt" (see below)

```
face-1-sects=01-normal=uncontrolled-abc=123456789.12-angs=keep-default
face-1-sects=01-normal=123456789.12-abc=keep-default-angs=default+edge
face-1-sects=01-normal=4-k-stations-abc=keep-default-angs=def-1-read-2
face-1-sects=01-normal=read-each-pt-abc=keep-default-angs=read-1-def-2
face-1-sects=01-normal=read-each-pt-abc=keep-default-angs=read-each-pt
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555556666666667
```

The face numbers should appear in numerical order, from one to six. The face may be divided into sections. The maximum number of sections per face is 10.

The overall purpose of the data in columns 24-35 is to specify whether or not the cell heights on the boundary surface will be controlled or not, if so how the required cell heights are to be given, and in one case to actually give the height. There are four different forms acceptable here:

- The first is "uncontrolled." This means that the control terms are deactivated on this face. This boundary treatment should be used for any boundary that is not a fixed boundary.
- The second form of input is to simply enter a floating-point number. This activates the control terms on this face. The program will try to make the cells touching this face be locally near-orthogonal, and will try to make them the height, given in user units, by the floating-point number. This input form causes the program to attempt to make the cell heights on this face be of uniform height.
- The third form of input in columns 24-35 is listed as "n-i-stations." The use of quotes around that datum is questionable, since that character-string as is should never be used. In place of the "n" a number from 2 to 9 should be substituted. In place of the "i" an index ("j" or "k" or "l") should be substituted. For certain problems the user might require cell heights on a face which are controlled, but are not uniform. 3DGRAPE/AL allows the specification of cell heights which are invariant with respect to one index but are varying as a piecewise continuous linear function of the other index. This form of input allows that. The piecewise continuous linear function is defined by giving the desired cell height at several values of the index, including its end points. The number in place of the "n" is the number of points at which a value for the cell height is to be given. The index substituted for the "i" is the index at values of which cell heights are to be given. For example, "4-k-stations" means that at k equals 1, at k equals its maximum

value, and at two intermediate values of k, cell heights will be given. The required cell heights between those places will be found by linear interpolation.

- Lastly, the data in columns 24-35 may be "read-each-pt". This gives the user total flexibility in setting the required cell heights. From some other source, such as a small program he has written just for this purpose, the user supplies one floating point number per point on the surface, giving the required heights of cells on the surface. The points and cells are ordered in the standard Fortran way, with the first subscript varying fastest. The cell heights should be a smoothly varying function over the boundary face. The user is responsible for making them so. If they are discontinuous, problems will result. These data are supplied in input file12. See the description of it in a subsequent section.
  Control should never be activated on a face which has coincident points. Where points are coincident, certain derivatives are undefined. The calculation of the S&S RHS terms requires all derivatives of first and second order. But for an error trap, division by zero would result.

A tip --> Realize that you don't have to use control terms everywhere; just use them on those boundary surfaces where you really care about the cell height. Let the elliptic method supply them elsewhere. This will simplify things and contribute to the robustness of the solution.

Another tip --> As you turn the control terms on there is a "game plan" you might want to follow. First, for each face having control terms activated, calculate the physical distance from a typical point on that face to its correspondent point the opposite face. That distance should then be divided by the number of intervals on the line connecting those points, yielding what would be the spacing on that line if that spacing was uniform. The user should compare that uniform spacing to the spacing being requested. For the first try, the requested spacing should be between one-half and one-fifth of the uniform spacing. Once convergence has been achieved, if you want smaller spacing at the wall you can then reduce the requested spacing in increments. Just how much it can be reduced is dependent on both the problem and the precision of your machine, and is impossible to predict generally. The symptom of not working, of course, is that the iterative grid-generation process will not converge. These multiple runs can either be repeated restarts, or each can start from initial conditions, at your discretion.

One more tip --> Consider the sizes of the grid cells on the boundary faces where control is activated. Divide the greatest dimension of any cell on the surface by the height being requested. It is recommended that that aspect ratio not be less than one, i.e., cells on the wall should not be taller than they are wide. For the first try, as in the preceding paragraph, that ratio should be no larger than about 5. Once that has worked you may increase that ratio in increments, by reducing the normal distance given in columns 24-35. Grids have been generated with aspect ratios as large as 10,000:1.

When control on a face is activated, 3DGRAPE/AL will attempt to make the grid cells immediately adjacent to that face conform to the required cell height and skewness. With distance from the face, into the interior of the block, control of height and skewness decays. Thus in the middle of the block the grid is essentially uncontrolled. That decaying control allows the distance between points on lines normal to the face to increase in a quasi-exponential manner with distance from the face. But how fast does that control decay with distance inward? There is a parameter, called abc, which influences the rate

of decay. The default value for that parameter is 0.45. The user may override that default by placing a floating-point number in columns 37-39. A larger number, such as 0.60 or 0.70, will cause the control to decay more rapidly, and will make the grid-generation convergence more stable. Decreasing that parameter to values such as 0.40 or 0.35 will cause the control to be propagated farther into the field, at the expense of decreasing the stability of the grid-generation convergence.

The "angs" input datum, in columns 59-70, specifies whether the grid lines intersecting the boundary surface are to be locally orthogonal, or not, and if not, then by just how much and in what way. There are five acceptable values. The first is "keep-default". This causes the program to attempt to make all the lines which intersect the boundary surfaces to do so orthogonally.

The second acceptable value is "default+edge". This is the paradigm which replaces the "lightening/tightening" feature in the earlier version of the code. It means that the program will attempt to make the lines intersect the boundary surface in a locally orthogonal manner everywhere except near a specified coordinate line (or lines) running across the surface. There it will bend the grid lines toward or away from the specified line in a manner which eases the grid over the discontinuity. It is expected that the user will use this in instances where the physical model being gridded has a sharp corner running across a surface. An example would be a cylindrical grid wrapping around an aircraft fuselage which has a strake.

The third and fourth acceptable values make reference to the fact that when a grid line intersects a boundary surface, to specify its orientation requires two data. Intersecting at every point on a grid boundary surface are two surface coordinate lines. There are two indices running on the surface, with the remaining index fixed on the surface. One running index varies along one of the coordinate lines, and the other running index varies along the other coordinate line. The six faces of the computational cube are arbitrarily given numbers, as shown in the following table. For purposes of this input datum we order the indices alphabetically, also shown:

Face number:	Fixed index:	First running index:	Second running index:
1	j=1	k	l
2	j=jmax	k	l
3	k=1	j	l
4	k=kmax	j	l
5	l=1	j	k
6	l=lmax	j	k

Table 7. Face Numbers and Indices

When "def-1-read-2" is given in columns 59-70 the angle that the line intersecting the surface makes with the first coordinate line (along which varies the first running index) is required to be 90°, while the angle the line makes with the other coordinate line (along which varies the second running index) is read in from file12. When "read-1-def-2" is given in columns 59-70 the angle that the line intersecting the surface makes with the first coordinate line (along which varies the first running index) is read in from file12, while the angle the line makes with the other coordinate line (along which varies the second running index) is required to be 90°.

The fifth acceptable value in columns 59-70 is "read-each-pt". When this is chosen, the angles with respect to both the first and second coordinate lines on the boundary surface are read from file12.

The "norm/sect" line

Line no.:	Column nos:	Datum type:	Description:
1	1-10	st	"norm/sect="
1	11-13	i	value of the index locating first point
1	14	st	"-"
1	15-26	f	cell height at first point
1	27	st	"-"
1	28-30	i	value of the index locating second point
1	31	st	"-"
1	32-43	f	cell height at second point
1	44	st	"-"
1	45-47	i	value of the index locating third point
1	48	st	"-"
1	49-60	f	cell height at third point

```
norm/sect=001-   3.       -011-  6.3       -020-  37.3
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555556
```

Whether there are other lines of input describing the face is dependent upon what values appear on the "face" line. The "norm/sect" line (or lines) should be present only if "n-i-stations" is chosen for columns 24-35 on the "face" line. It should immediately follow the "face" line, giving the values for cell height which make up the piecewise linear function. There may be up to three of these lines, allowing up to nine stations across a face. Subsequent of these lines have exactly the same format as the first.

The first type of "edges" line

Line no.:	Column nos:	Datum type:	Description:

1-2	c/i	"no", or the number of values of the first index at which there is an edge along which the second index runs
3	st	"-"
4	c	the first running index
5-11	st	"-edges-"
12-13	c/i	"no", or the number of values of the second index at which there is an edge along which the first index runs
14	st	"-"
15	c	the second running index
16-21	st	"-edges-"
22-28	st	"-nramp="
29-31	c/i	"kdf" or the number of points over which the edge treatment is to be ramped

```
no-j-edges-10-k-edges
no-j-edges-10-k-edges-nramp=kdf
no-j-edges-10-k-edges-nramp=005
12345678901234567890123456789012
         11111111112222222223333
```

Whether or not this line should be present depends upon what is given on the "face" line. It should be present only if columns 59-70 of the face line contain "default+edge". Above, the concept of sharp edges in the geometry being coincident with surface grid lines was introduced. If that is the case, and the special treatment for it is to be employed, this line must appear. It tells the program how many edges being coincident with each of the two families of surface coordinate lines there are.

Each of the example lines, immediately above, describe edges running across a face numbered 5 or 6. We know that because the running indices on that face are given as "j" and "k" (see the preceding table on face numbers and indices). The example input data record tells us that there are no values of j at which there is an edge having k running along its entire length. It also tells us that there are ten values of k at which there is an edge having j running along its entire length.

The last datum on this line, nramp, in columns 29-31, is the number of cells to each side of the sharp edge over which the special treatment is applied. It is "ramped up" from none to maximum right at the edge. The default value here, 5, will be used if "kdf," meaning "keep default" is entered. Other positive integers can be used instead.

The second type of "edges" line

Line no.:	Column nos:	Datum type:	Description:

1-9	st	"edges-at-"
10	c	the running index referred to in the preceding "edges" line
11	st	"="
12-14	i	a value of the index at a constant value of which the edge is
15	st	"-"
16-18	i	a value of the index at a constant value of which the edge is
19	st	"-"
20-22	i	a value of the index at a constant value of which the edge is

...and continuing across the line in the obvious way as needed

```
edges-at-k=001-007-017-044-065-176-280-335-399-401
1234567890123456789012345678901234567890123456789O
         1111111111222222222233333333334444444445
```

This line, like the previous (first type of) edge line, should appear only if columns 59-70 of the face line contains "default+edge". There should be one of these lines for each of the two indices which has edges, as described in the previous (first type of) edge line. In other words, the edge treatment can be applied along one index direction, or in both index directions; corresponding to that, there should be one or two of these lines. The example of the previous (first type of) edge line showed no edge treatments along lines having fixed j, and ten edge treatments along lines having fixed k. There are, in this example, 10 such lines of fixed k, at the given values of k, with j varying along those lines.

Summarizing now, there is an outside loop on the block number, and within that there is an intermediate loop on the face number. For each of the six faces, in numerical order, there must be a "face" line. Then, depending on whether or not they are called for in the "face" line, there may be "norm/sect" line(s) and the two types of "edge" lines. At this point all of the data pertaining to the face (as distinct from the sections into which it may be divided) have been read. It is time to go into the innermost loop on section number for each face. Typically, each face is one section. The ability to divide a face into multiple sections is rarely used, and so in what follows "section" can usually be thought of as equivalent to "face." However, when multiple sections are needed, that capability is available and very important.

It is for purposes of determining the X,Y,Z locations of the points on the boundary faces of the block that faces can be divided into sections. A face can be divided into as many as ten sections. There are eight different boundary treatments 3DGRAPE/AL offers for locating boundary points, and any of those treatments may be applied to each section.

The preceding input lines were given in the order in which they appear in the input file. But here ends any semblance of such order, since the boundary treatments listed below may be applied to any section of any face.

The following specifications for boundary treatment of sections of faces all include the range of indices to which those treatments apply. It is the user's responsibility to check those ranges to make sure that they add up to treatment of the entire face. It would be quite possible to divide a face into sections by index limits and leave holes untreated or have overlapping treatments. Overlapping treatments are inelegant, but rarely cause problems. Leaving holes untreated, however, should be avoided.

A closely related problem is treating the edges of the block, each of which is the intersection of two faces. Here again they might be treated once, by one of the two intersecting faces, twice, by both of the two intersecting faces, or they might be not treated at all. Redundant treatment is clumsy, but not a fatal error. When there are such redundancies, the treatment associated with the face having the highest face number will take precedence. But failing to treat an edge in any way will be a sure cause of failure. A checking procedure has been added to the code which checks for nonexistent or redundant treatment of edges and corners, and will give warnings or error messages if they are encountered.

Immediately following the input(s) pertaining to each face there should follow one of the following boundary treatment inputs for each section on the face, with no intervening blank lines. The ordering of the sections of each face is irrelevant.

The "read-in-fixed" line

Line no.:	Column nos:	Datum type:	Description:
	1-18	st	"read-in-fixed-xyz-"
	19	c	first index on the face: "j" or "k"
	20-25	st	"-from-"
	26-28	i	starting value of first index
	29-32	st	"-to-"
	33-35	i	ending value of first index
	36	st	"-"
	37	c	second index on the face: "k" or "l"
	38-43	st	"-from-"
	44-46	i	starting value of second index
	47-50	st	"-to-"
	51-53	i	ending value of second index

```
read-in-fixed-xyz-j-from-001-to-025-k-from-001-to-025
12345678901234567890123456789012345678901234567890123
         11111111112222222222333333333344444444445555
```

This treatment is used for inputting a fixed boundary surface, typically the shape or part of the shape about which or inside of which the user desires to make a grid. In other words, this treatment is used for the "body" of interest. As stated previously, these points on this surface must be distributed properly by some other device prior to input here. The points on this surface must be distributed with two running indices, as is typical of any surface mapping into the side of a computational cube. Those X,Y,Z data are not actually read from this file, file10. Instead, upon reading the "read-in-fixed" input line, 3DGRAPE/AL looks to file11 from which it actually reads the data. File11 is described in a subsequent section. After reading X,Y,Z data for this section of this face from file11 the program returns to file10 and continues reading.

The "plane-normal-to" lines

Line no.:	Column nos:	Datum type:	Description:
1	1-16	st	"plane-normal-to-"
1	17	c	axis to which perpendicular: "x" or "y" or "z"
1	18-26	st	"-axis-at-"
1	27	c	axis to which perpendicular: "x" or "y" or "z"
1	28	st	"="
1	29-40	f	location on axis
1	41	st	"-"
1	42	c	first index on the face: "j" or "k"
1	43-48	st	"-from-"
1	49-51	i	starting value of first index
1	52-55	st	"-to-"
1	56-58	i	ending value of first index
1	59	st	"-"
1	60	c	second index on the face: "k" or "l"
1	61-66	st	"-from-"
1	67-69	i	starting value of second index
1	70	st	"-"
2	1-6	st	"...to-"

2	7-9	i	ending value of second index
2	10-19	st	"-ext/proj="
2	20-31	f	the extrapolate/project parameter
2	32-47	st	"-initial-point=("
2	48-59	f	the value of the first coordinate at the initialization point
2	60	st	","
3	1-3	st	"..."
3	4-15	f	the value of the second coordinate at the initialization point
3	16	st	")"

```
plane-normal-to-y-axis-at-y=   17.98    -j-from-001-to-025-k-from-001-
...to-025-ext/proj=  0.0       -initial-point=(  4.0       ,
...     -2.5   )
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555556666666667
```

The points in this section, as defined by the given indices, will be constrained to lie on a plane normal to the indicated axis, at the indicated value on that axis. The distribution of points on that plane will be found by extrapolating from the elliptic grid solution in the interior of the block.

There are two different algorithms used for extrapolating to the plane from points in the interior. The first is a straight drop, from the neighboring point in the interior directly to the plane. This is the most stable, and is recommended for most applications. This type of extrapolation is selected by entering 0.0 in the "ext/proj" field, in columns 20-31 of the second line. The second algorithm extrapolates from three points in the interior by the use of a parabola. This method is specially designed to try and bring the point into the plane normally, to reduce the tendency for the method to go unstable. This method should be used only when an adjacent side boundary face does not intersect this face normally, and the user wishes to make this extrapolation more sensitive to the adjacent boundary shape. This type of extrapolation is selected by entering 1.0 in the "ext/proj" field, in columns 20-31 of the second line. The user can use a blending of the two methods by entering a number between 0.0 and 1.0 in this field.

Initially, all the points in this section are put at one point somewhere on the plane. With successive iterations the points spread out and go to where they should be. But that initial point could be anywhere on the plane, which is infinitely large. The user supplies the location of the initial point. Of the three coordinates X,Y,Z, one is constant on the plane and is specified on the first line. The other two coordinates, in alphabetical order, are given by the user in columns 48-59 on the second line and columns 4-15 on the third line. In this example, the plane is normal to the y axis at y=17.98, and therefore the user gives the X and Z coordinates of the initial point on the second and third lines as 4.0 and -2.5.

A tip --> The fact that points on the plane are initialized to the one initial point is one reason why the grid, when plotted after zero iterations (i.e., when set to the initial conditions) sometimes looks weird. But most of that weirdness goes away after the first few iterations.

The "cylinder-about" lines

Line no.:	Column nos:	Datum type:	Description:
1	1-15	st	"cylinder-about-"
1	16	c	name of axis: "x" or "y" or "z"
1	17-27	st	"-axis-from-"
1	28	c	name of axis: "x" or "y" or "z"
1	29	st	"="
1	30-41	f	starting value on axis
1	42-45	st	"-to-"
1	46	c	name of axis: "x" or "y" or "z"
1	47	st	"="
1	48-59	f	ending value on axis
1	60	st	"="
1	61	c	name of index along cylinder: "j" or "k" or "l"
2	62-68	st	"-along-"
2	1-13	st	"-axis-from-"
2	14-16	i	starting value of index along
2	17-20	st	"-to-"
2	21-23	i	ending value of index along
2	24	st	"="
2	25	c	name of index along cylinder: "j" or "k" or "l"
2	26-38	st	"-around-from-"
2	39-41	i	starting value of index along

2	42-45	st	"-to-"
2	46-48	i	starting value of index around
2	49-60	st	"-with-angle="
3	1-3	st	"..."
3	4-15	f	starting value of angle around (in degrees)
3	16-25	st	"-to-angle="
3	26-37	f	ending value of angle around (in degrees)
3	38-45	st	"-radius="
3	46-57	i	radius of cylinder
3	58-67	st	"-ext/proj="
4	1-3	st	"..."
4	4-15	f	the extrapolate/project parameter

```
cylinder-about-x-axis-from-x=100.        -to-x=     750.    -j-along-
...axis-from-002-to-033-1-around-from-002-to-021-with-angle=
...-90.        -to-angle= +90.       -radius=     500.    -ext/proj=
...  0.0
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

The points in this section, as defined by the given indices, will be constrained to lie on the surface of a cylinder. That cylinder must have its axis coincident with one of the coordinate axes. The program needs to know the limits of the cylinder in the axial direction. Note that the "starting value" on the axis should correspond to the starting value of the index running along the axis, and the "ending value" on the axis should correspond to the ending value of that index. The index limits should be given in increasing fashion, i.e., the ending limit of the index should be greater than the starting value. But the physical problem may demand that the values on the axis corresponding to those indices be given in decreasing fashion, i.e., the ending value on the axis may be smaller than its starting value. That is acceptable.

The cylinder need not displace the entire 360°. For example, in an aerodynamic application which assumes no yaw, the grid typically covers only one side, requiring a cylindrical section of 180°. Thus starting and ending values of the angle around the cylinder are input. Those angles are defined according to the increasing index convention for right-handed coordinate systems, and a decreasing index convention for left-handed systems. An alternate explanation of that angle definition is as follows. The cylinder's axis is one of the coordinate axes. The user should imagine his eye far out on the positive end of that axis, looking back toward the origin at the entire grid. The user will then be looking at a coordinate plane in which lie the two other axes. That plane should be rotated, and the entire grid with it, about the cylindrical axis until the positive end of one of those other two axes points to the right and the other positive end points up. The user

can then imagine a conventional 2-D polar coordinate system on that plane, with the angle equal to zero on the right and increasing in counterclockwise fashion. It is with respect to that angle that the starting and ending angles entered in columns 4-15 and 26-37 of the third input line are measured.

The axis values and the angles are used only for locating the initial conditions. Thus great precision is not required.

Note that the starting and ending values of the index running around the axis should be given in increasing order, i.e., the ending value must be greater than the starting value. But the starting and ending values of the angle need not be so ordered; the physical problem may require that they be ordered backwards. That is acceptable.

As with the plane-normal-to, above, there are two algorithms available to extrapolate to the cylinder from points inside of it. The first, selected by using 0.0 for "ext/proj=", does a simple projection from the point inside to the nearest point on the cylinder, i.e., to a point at the same axial station, the same angle around the axis, and a greater radius. Using 1.0 for "ext/proj=" causes the point to be extrapolated linearly from two points inside of the cylinder. The user can blend the two treatments by using a value between 0.0 and 1.0. Use of 0.0 is recommended.

The "ellipsoid" line

Line no.:	Column nos:	Datum type:	Description:
1	1-17	st	"ellipsoid-x-cent="
1	18-29	f	x-coordinate of center of ellipsoid
1	30-37	st	"-y-cent="
1	38-49	f	y-coordinate of center of ellipsoid
1	50-57	st	"-z-cent="
1	58-69	f	z-coordinate of center of ellipsoid
2	1-10	st	"...x-semi="
2	11-22	f	length of semi-span in x-direction
2	23-30	st	"-y-semi="
2	31-42	f	length of semi-span in y-direction
2	43-50	st	"-z-semi="
2	51-62	f	length of semi-span in z-direction
2	63	st	"-"
2	64	c	name of first index: "j" or "k"
2	65-70	st	"-from-"

3	1-3	st	"..."
3	4-6	i	starting value of first index
3	7-10	st	"-to-"
3	11-13	i	ending value of first index
3	14	st	"-"
3	15	c	name of second index: "k" or "l"
3	16-21	st	"-from-"
3	22-24	i	starting value of second index
3	25-28	st	"-to-"
3	29-31	i	ending value of the second index
3	32-55	st	"-latitudinal-angle-from-"
3	56-67	f	starting latitudinal angle
3	68	st	"-"
4	1-6	st	"...to-"
4	7-18	f	ending latitudinal angle
4	19-30	st	"-polar-axis="
4	31	c	"x", "y", or "z", the polar axis
4	32-38	st	"-index="
4	39	c	"j", "k", or "l", the index which goes along the axis
4	40-64	st	"-longitudinal-angle-from-"
5	1-3	st	"..."
5	4-15	f	starting longitudinal angle
5	16-19	st	"-to-"
5	20-31	f	ending longitudinal angle
5	32-41	st	"-ext/proj="
5	42-53	f	the extrapolate/project parameter

```
ellipsoid-x-cent=100.        -y-cent=   0.       -z-cent=   0.
...x-semi=  500.      -y-semi=   500.     -z-semi=  500.      -j-from-
...002-to-018-1-from-002-to-021-latitudinal-angle-from-123456789.12-
...to-123456789.12-polar-axis=x-index=j-longitudinal-angle-from-
...123456789.12-to-123456789.12-ext/proj=123456789.12
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666 7
```

The points on a face, or on a section of a face, may be constrained to lie on the surface of an ellipsoid. A sphere, of course, is a special case of an ellipsoid. The center of the ellipsoid may lie anywhere, and that location is given on the first line. The ellipsoid must, however, have its semi-axes parallel with the coordinate axes. The shape of the ellipsoid is defined by the length of the semi-axes. The length of the semi-axis in the x-direction, i.e., the distance from the center to the surface measured in the x-direction, is given in columns 11-22 of the second line. The other semi-axes are given similarly.

We will initialize the points on this ellipsoid as lines of latitude and longitude. For purposes of initialization the user designates which axis is to be the polar axis, and the range of latitudinal angles (north and south of the equator on the earth). Positive angles are toward the positive end of the given polar axis, zero is at the equator, and negative angles are toward the negative end of the given polar axis. Thus these angles must be between +90 and -90 degrees. We also need to know which index goes in the latitudinal direction. This index goes from the "from" latitudinal angle to the "to" latitudinal angle. Thus by swapping these angles the index can be made to run the opposite way.

We also need to know the range of longitudinal angles (east and west on the earth). Those angles are defined according to the increasing index convention for right-handed coordinate systems, and a decreasing index convention for left-handed coordinate systems. This is the same way as the angle in the cylindrical face treatment. The user should imagine his eye far out on the positive end of the polar axis, looking back toward the entire grid. The user will then be looking at a coordinate plane in which lie the two other axes. That plane should be rotated, and the entire grid with it, about the polar axis until the positive end of one of those other two axes points to the right and the other positive end points up. The user can then imagine a conventional 2-D polar coordinate system on that plane, with the longitudinal angle equal to zero on the right and increasing in counterclockwise fashion.

For example, suppose that in a right-handed coordinate system the Y axis is given as the polar axis. This leaves the X and Z axes in the plane. Rotating them as described above results in the positive end of the X axis being to the right and the Z axis pointing upward. Thus, in this case, the longitudinal angle would be measured from the X axis around counterclockwise, encountering the positive end of the Z axis at 90 degrees.

Because we know which face number this is we know which two indices run over it. Thus, knowing which index runs in the latitudinal direction, we, by process of elimination, know which index runs in the longitudinal direction. That index will be equal to its minimum on this section at the "from" longitudinal angle, and at its maximum at the "to" longitudinal angle. As above, this index may be caused to run the other way by swapping the "from" and "to" angles.

The user should realize that all the foregoing about locating the points at lines of constant latitude and longitude applies only to the initialization. The points are free to move around during the iteration process, according to the emerging solution in the interior of the block.

As with the plane-normal-to, above, there are two algorithms available to extrapolate to the ellipsoid from points inside of it. The first, selected by using 0.0 for "ext/proj=", does a simple projection from the point inside to the nearest point on the ellipsoid. Using 1.0 for "ext/proj=" causes the point to be extrapolated linearly from two points inside of the ellipsoid. The user can blend the two treatments by using a value between 0.0 and 1.0. Use of 0.0 is recommended.

The "collapsed-to-an-axis" lines

Line no.:	Column nos:	Datum type:	Description:
1	1-13	st	"collapsed-to-"
1	14	c	name of the axis: "x" or "y" or "z"
1	15-25	st	"-axis-from-"
1	26	c	name of the axis: "x" or "y" or "z"
1	27	st	"="
1	28-39	f	starting value on the axis
1	40-43	st	"-to-"
1	44	c	name of the axis: "x" or "y" or "z"
1	45	st	"="
1	46-57	f	ending value on the axis
1	58	st	"-"
1	59	c	name of index along axis: "j" or "k" or "l"
1	60-66	st	"-along-"
2	1-13	st	"...axis-from-"
2	14-16	i	starting value of the index along axis
2	17-20	st	"-to-"
2	21-23	i	ending value of the index along axis
2	24	st	"-"
2	25	c	name of index around axis: "j" or "k" or "l"
2	26-38	st	"-around-from-"

2	39-41	i	starting value of index around axis
2	42-45	st	"-to-"
2	46-48	i	ending value of index around axis
2	49-58	st	"-ext/proj="
2	59-70	f	the extrapolate/project parameter

```
collapsed-to-x-axis-from-x=    0.      -to-x= -400.      -k-along-
...axis-from-002-to-031-1-around-from-001-to-022-ext/proj=   0.0
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

Certain topologies, such as spherical or cylindrical grids, give rise to the need for a face, or a section of a face, to be collapsed to an axis. This input option allows that treatment. Note that the points on the axis are found by extrapolating to the axis, and so the distribution of points on the axis is that which results from the elliptic solution. Elliptic grids tend to be uniformly distributed, absent the effect of control terms. Thus the distribution of points on faces collapsed to axes tends to be uniform.

The axis values given here are used only for locating the initial conditions. Thus great precision is not required.

The starting and ending values of the indices should be given in increasing order, i.e., the ending values should be larger than the starting values. This sometimes means that the corresponding starting and ending values on the axis must be given in decreasing order, i.e., with the ending values less then the starting values. That is acceptable.

There are two different algorithms used for extrapolating to the axis from points in the interior. The first is a straight drop, from the neighboring point in the interior directly to the axis. This is the most stable, and is recommended for most applications. This type of extrapolation is selected by entering 0.0 in the "ext/proj" field. The second algorithm extrapolates from three points in the interior by the use of a parabola. This method is specially designed to try and reduce the tendency for the method to go unstable. This method should be used only when an adjacent side boundary face does not intersect this face normally, and the user wishes to make this extrapolation more sensitive to the adjacent boundary shape. This type of extrapolation is selected by entering 1.0 in the "ext/proj" field. The user can use a blending of the two methods by entering a number between 0.0 and 1.0 in this field.

As faces collapsed to axes have many coincident points, control terms should not be activated thereon.

The "collapsed-to-a-point" lines

Line no.:	Column nos:	Datum type:	Description:
1	1-21	st	"collapsed-to-point-x="
1	22-33	f	x-coordinate of the point

1	34-36	st	"-y="
1	37-48	f	y-coordinate of the point
1	49-51	st	"-z="
1	52-63	f	z-coordinate of the point
1	64-69	st	"-with-"
2	1-3	st	"..."
2	4	c	name of first index: "j" or "k"
2	5-10	st	"-from-"
2	11-13	i	starting value of first index
2	14-17	st	"-to-"
2	18-20	i	ending value of first index
2	21	st	"-"
2	22	c	name of second index: "k" or "l"
2	23-28	st	"-from-"
2	28-31	i	starting value of second index
2	32-35	st	"-to-"
2	36-38	i	ending value of second index

```
collapsed-to-point-x=  750.      -y=   0.       -z= 0.          -with-
...j-from-001-to-001-1-from-001-to-022
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666667
```

Because all points on this section are coincident, control must not be activated here.

The "match-to-face" lines

Line no.:	Column nos:	Datum type:	Description:
1	1-14	st	"match-to-face-"
1	15	i	face number of other face

1	16-22	st	"-block-"
1	23-24	i	block number of other face
1	25-30	st	"-this-"
1	31	c	name of first index on this face: "j" or "k"
1	32-37	st	"-from-"
1	38-40	i	starting value of first index on this face
1	41-44	st	"-to-"
1	45-47	i	ending value of first index on this face
1	48-53	st	"-this-"
1	54	c	name of second index on this face: "k" or "l"
1	55-60	st	"-from-"
1	61-63	i	starting value of second index on this face
1	64-67	st	"-to-"
1	68-70	i	ending value of second index on this face
2	1-9	st	"...-that-"
2	10	c	first index on that face: "j" or "k" or "l"
2	11-16	st	"-from-"
2	17-19	i	starting value of first index on that face
2	20-23	st	"-to-"
2	24-26	i	ending value of first index on that face
2	27-32	st	"-that-"
2	33	c	second index on that face: "j" or "k" or "l"
2	34-39	st	"-from-"
2	40-42	i	starting value of second index on that face
2	43-46	st	"-to-"
2	47-49	i	ending value of second index on that face
2	50-64	st	""

3	1-4	st	"...("
3	5-16	f	X coordinate of the initial point
3	17	st	","
3	18-29	f	Y coordinate of the initial point
3	30	st	","
3	31-42	f	Z coordinate of the initial point
3	43	st	")"

```
match-to-face-1-block-02-this-k-from-002-to-031-this-l-from-001-to-022
...-that-k-from-002-to-031-that-l-from-001-to-022-initial-point=
...(   3.        ,   5.        ,  -2.7       )
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667
```

This boundary treatment allows this section of this face to be matched to (1) any other section of this face, or (2) any section of another face of this block, or (3) any section of any face of any other block. That match will produce a block-to-block type boundary where the surface floats with the solution of the grid-generation equations. Grid line slope and spacing will be continuous across this surface. Note that this surface is double-stored, i.e., it exists in memory identically as part of both coincident faces.

The range of indices defining "this" section must match with the range of indices defining "that" section. Note that while the first index on "this" face must be j or k and its second index must be k or l, any index could be the first index on "that" face and any other index could be its second index. The starting and ending values on "this" face must be given in increasing fashion, i.e., the ending values must be greater than the starting values. But the corresponding indices on "that" face may run in whatever direction is appropriate. A checking procedure has been introduced which checks the match-to-face data for consistency. It prints warnings and error messages if appropriate. Note that the information given here must essentially be given twice—once here in these input lines describing "this" face of "this" block, and also in the input lines describing "that" face of "that" block.

As with the plane-normal-to boundary treatment, there is an initial point given here. All the points on this face will be initialized to this value, which looks very strange when plotted. This condition will be corrected after the first iteration. Note that this initial point requires all three coordinates.

The "freeze-at-restart" line

Line no.:	Column nos:	Datum type:	Description:
	1-18	st	"freeze-at-restart-"
	19	c	first running index on this face

20-25	st	"-from-"
26-28	i	starting value of first running index
29-32	st	"-to-"
33-35	i	ending value of first running index
36	st	"-"
37	c	second running index on this face
38-43	st	"-from-"
44-46	i	starting value of second running index
47-50	st	"-to-"
51-53	i	ending value of second running index

```
freeze-at-restart-j-from-001-to-025-k-from-001-to-025
12345678901234567890123456789012345678901234567890123
         11111111112222222222333333333344444444445555
```

This new boundary treatment may be used at the time of a restart, to freeze all the points on a boundary surface. It is intended for use on sections which in previous runs were floating -- on a plane, cylinder, ellipsoid, or axis.

FILE11 – BODY DEFINITION ARRAYS

It was stated in the previous section that during its input phase 3DGRAPE/AL reads through file10 until it encounters a "read-in-fixed" input line. At that point it suspends reading from file10 and begins reading the fixed surface points from file11. When it is finished reading those X,Y,Z coordinates from file11 it returns to reading from file10. This cycle will be repeated as many times as there are "read-in-fixed" input lines. Thus file11 must contain X,Y,Z coordinates of as many fixed surfaces as there are "read-in-fixed" input lines.

For each read-in-fixed-surface file11 must contain:

- a header line introducing the x-coordinates,
- the x-coordinates,
- a header line introducing the y-coordinates,
- the y-coordinates,
- a header line introducing the z-coordinates, and
- the z-coordinates.

No intervening blank lines are allowed. This cycle of six things should be repeated for each fixed surface.

The header lines introducing the coordinates are of the form:

Line no.:	Column nos:	Datum type:	Description:
	1-9	st	"complete-"
	10	c	name of coordinate: "x" or "y" or "z"
	11-23	st	"-for-section-"
	24-25	i	section number
	26-34	st	"-of-face-"
	35	i	face number
	36-45	st	"-of-block-"
	46-47	i	block number
	48-49	st	"-f"
	50-51	c	field width, either "12" or "20"

```
complete-x-for-section-01-of-face-3-of-block-01
complete-x-for-section-01-of-face-3-of-block-01-f12
complete-x-for-section-01-of-face-3-of-block-01-f20
12345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444455
```

As described immediately below, the X,Y,Z, data in this file may be in either 6f12.0 format or 4f20.0 format. Blanks or "-f12" on the end of the first header line in the file select 6f12.0;"-f20" on the end of the first header line in the file selects 4f20.0 The entire file must use the same one of those two formats, so this selection applies to the entire file. The contents of columns beyond 47 are optional on the first of these header lines in this file, and ignored on all subsequent header lines in this file.

The actual X- or Y- or Z-coordinates begin on the line immediately following their respective header line. As with all F formats used for input in Fortran, the placement of decimal points in the input record overrides the placement of decimal points as indicated on the format, and so the ".0" in the format is irrelevant. The selected format is repeated for subsequent lines as many times as needed. The points should be ordered with the first running subscript varying most rapidly as the "inner loop," and the second running subscript varying most slowly as the "outer loop."

FILE12 - CELL HEIGHTS AND ANGLES AT BOUNDARY SURFACES

Whenever called for by the "normal=" and/or "angs=" fields in the "face" lines, the program will suspend reading from file10 and begin reading from file12. When finished

reading those data for that face from file12, it returns to reading from file10. The data in file12 appear in whatever order they are called for in the various "face" lines in file10.

Line no.:	Column nos:	Datum type:	Description:
	1-33	st	"complete-normal-heights-for-face-"
	34	i	face number
	35-44	st	"-of-block-"
	45-46	i	block number
	48-49	st	"-f"
	50-51	c	field width, either "12" or "20"

```
complete-normal-heights-for-face-3-of-block-08
complete-normal-heights-for-face-3-of-block-08-f12
complete-normal-heights-for-face-3-of-block-08-f20
12345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445
```

Line no.:	Column nos:	Datum type:	Description:
	1-30	st	"complete-angles-wrt-subscript-"
	31	i	running subscript number, either 1 or 2
	32-41	st	"-for-face-"
	42	i	face number
	43-52	st	"-of-block-"
	53-54	i	block number
	55-56	st	"-f"
	57-58	c	field width, either "12" or "20"

```
complete-angles-wrt-subscript-1-for-face-3-of-block-08
complete-angles-wrt-subscript-2-for-face-3-of-block-08-f12
complete-angles-wrt-subscript-2-for-face-3-of-block-08-f20
12345678901234567890123456789012345678901234567890123456789012345678
         11111111112222222222333333333344444444445555555555
```

The "normal-heights" header line is followed by the requested cell heights, in the user's own units as used in the X,Y,Z. They are ordered as are the X,Y,Z in file11, with the first running subscript varying most rapidly as the "inner loop," and the second running subscript varying most slowly as the "outer loop." The format is either 6f12.0 or 4f20.0, selected just as in file11.

The "angles" header line is followed by the requested angles, in degrees, with 90.0 meaning perpendicular, angles less than 90.0 tilting toward the end of the indicated surface coordinate with lower running index values, and angles more than 90.0 tilting toward the end of the indicated surface coordinate with higher running index values. "Wrt" stands for "with respect to."

FILE13 – TO READ IN A GRID AND SMOOTH IT

This program has the ability to read in a grid generated elsewhere, smooth it, and write it out again. With the graphical version of this program, such a grid could be read in, looked at, and then smoothed or left un-altered.

File13 is the logical unit used for reading in this grid. The name of this file is read from file19, described in a subsequent section. The grid file itself, file13, may be of any of the same forms in which this program writes file14, its output grid. These forms are described above, in the section called "the filename-14" line. This file to be read should have been written by code being functionally equivalent to the pseudo-code shown in that section.

FILE16 – CONTROL SCALARS FOR RE-START

As was said above, 3DGRAPE/AL has a restart capability. The user can let the grid generator run a while, examine the resulting grid, change some things, and then run it some more. The code allows the user to change many things at restart. The number of blocks, the dimension sizes of the blocks, the "handedness" of the blocks, the data pertaining to Cartesian vs. spherical topology, and whether or not control is activated on each face are the only things which cannot be changed. This means that all the other parameters originally given in file10 may be modified, the body shape given in file11 or the distribution of points on it may be modified, etc.

If a run of 3DGRAPE/AL is to be restarted, the first requirement is that a restart file be written as file15 by the first run; see the "write-for-restart" input line in file10, described above. For the restart run, this same file is read in as file17.

As described above, in the section called, "THE FIRST TWO LINES," the first thing the program does in any execution is to inquire about what type of run this is. For a restart the user should enter "re-start". In this restart case, the program reads a file, using unit 16, which bears great resemblance to a file10. In fact, most users create their file16 by copying their file10 and then suitably modifying it. The program will ask what filename is to be used for the file16 input data. The user should enter that name.

The following table lists the input lines in file16. Where it says "just like in file10" it does not mean that the line in file16 must be identical to the corresponding line in the file10 used in the first run for this grid; it means that the user has all the options in making up this line which are available for this line in any file10. However, in practice, most users will cause most of these lines to be the same as the corresponding line in their file10.

Line designation:	Description:
The "run-comment" lines	Just like in file10.

The "filename-17-input" line	This is a new type of line, present only in file16. It simply tells the program what filename is used for the restart file, written as file15 by the previous run. Its format is:
	
	`filename-17-input=my_restart_file`
	`123456789012345678901234567890123456`
	`1111111111222222222233333`

The "heading" line	This is a shortened version of the "number-of-blocks" line.
	
	`heading=kdf`
	`heading=000`
	`heading=054`
	`heading=kdf-filename-18-views=my_picture_data`
	`123456789012345678901234567890123456789012345`
	`1111111111222222222233333333333444444`

The "iterations" lines	Just like in file10, except that: • In restart runs the RHS types "S&S-initzero" and "S&S-init-T&M" make no sense, because in a restart the RHS are not being initialized. • In restart runs coarse parts are forbidden
The "filename-11" line	Just like in file10.
The "filename-14" line	Just like in file10.
The "write-for-restart" line	Just like in file10. However, you should realize that you can make multiple, subsequent restart runs if you want. If you don't want to do another restart after this one, be sure to say "no". But if you do, be careful with the filenames. In that case, make sure that you change the name of the restart file to be created on this run, so that you don't overstore the restart file you just read for this run.
The "omegpqr" line	Just like in file10.
The "quality-check" line	Just like in file10.
The "block-comment" line	Just like in file10.
The "dimension" line	This line must be removed. If anything on it changed, chaos would result.
The "handedness" line	This line must be removed. If anything on it changed, chaos would result.
The "polar-axis" line	This line must be removed. If anything on it changed, chaos would result.

The "freezeblock" line	Just like in file10.
The "face" line	Just like in file10.
The "norm/sect" line	Just like in file10.
The first type of "edges" line	Just like in file10.
The second type of "edges" line	Just like in file10.
The "read-in-fixed" line	Just like in file10.
The "plane-normal-to" lines	Just like in file10.
The "cylinder-about" lines	Just like in file10.
The "ellipsoid" line	Just like in file10.
The "collapsed-to-an-axis" lines	Just like in file10.
The "collapsed-to-a-point" lines	Just like in file10.
The "match-to-face" lines	Just like in file10.
The "freeze-at-restart" line	This option is available at restart, although it is not available on a first run.

Table 8. List of Input Lines Used in File16 Input

FILE18 – INDICES OF SURFACES TO BE VIEWED

The batch (non-graphical) version of this code ignores file18. The graphical version, however, reads file18 to find exactly what the user wants to look at when viewing the grid. The graphical version of the code allows the user to choose between as many as eight different views of the grid. As used here, the term "view" means a set of coordinate surfaces (over which one index is fixed, and the other two vary) or portions thereof. These coordinate surfaces can be taken from any or all of the blocks in a multiple-block grid. Each surface can be colored with any of eight colors. The background is black, unless all the surfaces are chosen to be black, in which case the background is white. The number of surfaces per view is limited by a parameter set in params.h. As delivered, this value is 50.

All of the data in file18 are read in 8i5 format. There is one "group" of data for each view, with the groups appearing in sequence. Each group begins with a line having on it one number: the number of surfaces in this view:

Line no.:	Column nos:	Datum type:	Description:
	1-5	i	number of surfaces in this view.

```
    3
12345678901234567890123456789012345678901
          11111111112222222222333333333334
```

In this example, the line, the first line of a group, says that the group consists of three grid surfaces.

The remainder of the group consists of one line of data for each surface in the view, defining the surface:

Line no.:	Column nos:	Datum type:	Description:
	1-5	i	Block number of this surface
	6-10	i	Minimum value of the first index, j, in this surface
	11-15	i	Maximum value of the first index, j, in this surface
	16-20	i	Minimum value of the second index, k, in this surface
	21-25	i	Maximum value of the second index, k, in this surface
	26-30	i	Minimum value of the third index, l, in this surface
	31-35	i	Maximum value of the third index, l, in this surface
	36-40	i	Color code for this surface, chosen from the Color Codes table, shown below

```
    1     1    50     1     1     1    40     3
    1     1    50    30    30     1    40     3
    1    25    25     1    30     1    40     5
12345678901234567890123456789012345678901234567890
          11111111112222222222333333333 4
```

Suppose the example lines, above, were used for viewing a one-block grid having index limits of 50 x 30 x 40. Then the first line describes a yellow surface where k is fixed at its minimum value, 1, with j and k running over their full ranges. The second line describes another yellow surface where k is fixed at its maximum value, 30, with j and k running over their full ranges. The third line describes another surface, magenta in color, where j is fixed at an intermediate value of 25, with k and l running over their full ranges.

Since each line of data after the first in each group represents a grid surface, and a grid surface by definition has one index fixed, that line of data should have one minimum index value equal to the maximum value of that same index. See k in the first and second lines, and j in the third line. This rule can be violated at the user's discretion (e.g., for drawing both upper and lower surfaces of a wing), but the user should realize that drawing solid figures (as opposed to surfaces) tends to produce an impenetrable blob, and takes the computer a long time to draw.

In a multiple-block grid surfaces from multiple blocks can be mixed in a view (in a group).

Blank lines may appear in this file preceding the lines which give the number of surfaces in each view. I.E., blank lines may appear between the groups of data. Using them enhances readability of the file. Blank lines may not appear within the groups of data.

This file is read until terminated by:

- having read eight views (eight groups of data), or
- encountering an end of file, or
- reading a negative number for the number of surfaces in a view. This allows other data, not currently in use, to be stored in the same file after the terminating negative number. Users have found this convenient.

Color code:	Resulting color:
0	black
1	red
2	green
3	yellow
4	blue
5	magenta
6	cyan
7	white

Table 9. Color Codes

FILE19 – CONTROL SCALARS FOR SMOOTHING A GRID

The control scalars for reading in a grid to be smoothed are read from file 19, and are similar to the data read from files 10 and 16.

Line no.:	Column nos:	Datum type:	Description:
	1-20	st	"run-comment "
	21-70	c	free-field comment describing this run

```
run-comment          Blah, blah, blah.
run-comment          What this data is all about.
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667
```

These two lines are just like in file10, above.

Line no.:	Column nos:	Datum type:	Description:
1	1-11	st	"iterations="
1	12-14	i	the number of iterations in this part
1	15-23	st	"-control="
1	24-25	c	overriding global switch on control, either "ye" or "no"

```
iterations=100-control=no
iterations=100-control=ye
12345678901234567890123 45
         1111111111222222
```

The data on this line are the same as described in file10.

Line no.:	Column nos:	Datum type:	Description:
	1-23	st	"filename-13-grid-input="
	24-38	n	filename for input on unit 13
	39-44	st	"-form="
	45-51	c	"3dgrape" or "plot3ds" or "plot3dm" or "charact"

```
filename-13-grid-input=my_bumpy_grid  -form=3dgrape
filename-13-grid-input=what_a_mess    -form=plot3ds
filename-13-grid-input=I_should_smooth-form=plot3dm
filename-13-grid-input=this_rascal    -form=charact
12345678901234567890123456789012345678901234567890123456789012345678901
         1111111111222222222233333333334444444445 5
```

The data on this line are the same as described in file10.

Line no.:	Column nos:	Datum type:	Description:
	1-24	st	"filename-14-grid-ouput="

25-39	n	filename for output on unit 14
40-45	st	"-form="
46-52	c	"3dgrape" or "plot3ds" or "plot3dm" or "charact"

```
filename-14-grid-output=my_smooth_grid -form=3dgrape
filename-14-grid-output=a_joy_to_behold-form=plot3ds
filename-14-grid-output=what_a_great   -form=plot3dm
filename-14-grid-output=program_this_is-form=charact
12345678901234567890123456789012345678901234567890123456789012
         111111111122222222223333333333444444444455
```

The data on this line are the same as described in file10.

Line no.:	Column nos:	Datum type:	Description:
	1-14	st	"quality-check="
	15-16	c	"ye" or "no"
	17-35	st	"-filename-18-views="
	36-50	n	name of file for input as file18

```
quality-check=ye
quality-check=no-filename-18-views=my_picture_data
123456789012345678901234567890123456789012345678901234567890
         111111111122222222223333333333444444444445
```

The data on this line are the same as described in file10.

OUTPUT

There are two principal forms of output from this program. The first is the "printout" file, i.e., the text which is written to standard output. It consists primarily of

- a listing of the control scalar input (file10 or file16 or file19), with some comments added about what the program is doing,
- a trace of the iteration count as the program runs,
- a convergence history for each block, and
- quality-check data, if called for.

The other principal output is file14, containing the finished grid. The various forms of this file are described in the section on "The filename-14" line, above.

The program also, if called for, makes a restart file (file15) to be read (as file17) by the program itself in the case of a restart. Since the user has no cause to examine this unformatted file, its format is not documented here.

One might also consider the pictures on the screen drawn by the graphical version of this code to be output. Furthermore, the plotting package can write screen-dump files of grid pictures named plotit.01.rgb, plotit.02.rgb, etc., and screen-dump files of convergence history plots named convhist.01, convhist.02, etc.

RUNNING THE GRAPHICAL USER INTERFACE

As was said above, when the program is run on a Silicon Graphics IRIS workstation it can be linked in two different versions. One is batch and the other is with graphics. This section will describe running the code in its graphics version.

A sample of the Graphical User Interface can be seen in the first grid figure, Figure 3a, appearing in the following section on the example cases. Those figures are reduced to grayscale in this manual, but liberal use is made of color in the actual program.

When the graphics opens there will be four windows on the screen. The big square one on the left displays the grid. On the right are three small windows.

The top small window is titled TRANSFORMS. The user should realize that although no axes are actually drawn, conceptually there are two different sets of axes here. One set is the body's axes, which move with the object. If the grid is about an airplane those body axes would be the pitch, roll, and yaw axes. The other set of axes is the screen axes, which are fixed. The screen axes go right and left, up and down, and in and out (which is zooming). This window's buttons give the user the ability to rotate the plotted object about the body's axes, translate it along those axes, and translate it along the screen's axes.

Once the use of the mouse and its buttons is understood, the rest is easy. In the TRANSFORMS window the user first uses the middle mouse button to select what operation is to be performed, i.e., to select which of the nine transforms is to be currently active. To activate a certain transform, place the mouse cursor over that screen button and click the middle mouse button. That screen button will then light up, signifying that that transform is active.

Then, with the desired transform selected, push the right and left mouse buttons to operate it. The right mouse button causes the transform to operate one way, and the left mouse button causes it to operate the other way. These transforms work only when the mouse cursor is somewhere in this window.

There is a default transform speed which is relative to the physical size of the object. But that speed isn't always right. And so the user can vary the speeds of the transforms, a different speed for each. Notice the little green triangles in each transform button. They indicate the speed of that transform -- faster to the right and slower to the left. The user moves them by moving the mouse to the right or left while actually doing the transform, i.e., by sliding the mouse to the right or left while actually depressing either the right or left mouse button.

The middle small window is titled CONTROLS. It offers variety of different control actions. In this window, and in the one below it, the middle mouse button selects what the user wants to do, and the right and left mouse buttons don't do anything.

The screen buttons in the CONTROLS window are described below:

Button:	Function:
GO	The program will automatically pause for in-depth graphical inspection at the start and at the end of the run. In addition, as described below, the user can pause the iterative process whenever he wants to. When finished inspecting the user should hit this GO button to resume iterating. Note that here, and throughout the other buttons, whenever a button is not available it will be dimmed. For example, after pushing GO the only button available is PAUSE.
PAUSE	The user can pause the program anytime during the iterative process. Just press and hold this button until the current iteration is finished.
EXIT	What it says. There is a "do you really want to do this?" trap.
RESET	It is possible, after many transform operations, to get lost. The user either can't tell where he is in the grid, or the entire picture is off of the screen and can't be found. This button resets the orientation of the object to what it was at the start.
FAST / SLOW	As described above, there is a speed setting for each of the transforms, with the current speed settings illustrated by the little green triangles in the TRANSFORMS window. This button cause all the triangles to toggle together to either the right -- FAST -- or to the extreme left -- SLOW.
PERS / ORTH	Toggle between perspective and orthographic projections. Some zooming may be required to make the object the same size after switching projections.
Z-BUFFER	This invokes hidden-surface removal. Just push the button. The program re-draws the current picture in hidden surface mode. Then hit anywhere in the CONTROLS window to get it back into normal mode.
PIXSAVE	This button causes the program to call the SGI utility "scrsave" to dump a raster image of the screen to a file of the RGB type. This is useful for making pictures and viewgraphs. Beware that in every run it names the files "plotit.01.rgb", "plotit.02.rgb", etc. Therefore, if there are files thusly named remaining from an earlier run they will be overwritten.
CONV HIST	This button causes the GUI to switch from plotting the grid to plotting convergence histories.

Table 10. GUI Control Buttons

When in CONV HIST mode, for each block there is either one or two convergence history plots: a plot showing maximum and average point movement vs. iteration count, and if S&S RHS terms are used a plot showing maximum absolute value of the RHS terms and maximum correction of RHS terms. These one or two plots are repeated for each block, making a list of convergence history plots. The convergence history plot

window shows, along with a convergence history plot, four buttons. Pushing them has the following effects:

Button:	Function:
NEXT	The next plot in the list will be shown.
PREVIOUS	The previous plot in the list will be shown.
PIXSAVE	A screen dump to a file will be performed.
RETURN	Return to grid plotting mode.

Table 11. Screen Buttons in CONV HIST Mode

By hitting one of the buttons in the bottom small window titled SELECT VIEWS with the middle mouse button the user selects between the different views which were pre-defined in the file18 views file.

EXAMPLE CASES

Three basic example cases are included, plus two variations on the first, for a total of five cases. They illustrate the use of many of the features of the code. Following is a table which lists the files relating to the example cases. They are all ASCII text. This table could be thought of as a continuation of Table 3, above.

File number:	Sub-directory:	File name:	Data in this file:
160	box	boxall.f10	Control scalars for the basic box case
161	box	boxall.f11	X,Y,Z coordinates of points on the sides of the box, having sinusoidal bumps. Used for all the box cases.
162	box	boxall.f18	Indices defining the grid surfaces to be viewed when any of the box cases are run in the graphical version.
163	box	boxall.out	Output file (printout file) for the basic box case
164	box	smooth.f19	Control scalars to read in the basic box case and smooth it a little. This data file is included on the tape, but this excercise is not listed in this manual as a test case.
165	wing	wing.f10	Control scalars for the wing with flat-plate extension case
166	wing	wing.f11	X,Y,Z coordinates of points on the wing with flat-plate extension case

167	wing	wing.f18	Indices defining the grid surfaces to be viewed when the wing with flat-plate extension case is run in the graphical version
168	wing	wing.out	Output file (printout file) for the wing with flat-plate extension case
169	hcc	hemcylcon.f10	Control scalars for starting the hemisphere-cylinder-cone case
170	hcc	hemcylcon.f11	X,Y,Z coordinates of points on the hemisphere-cylinder-cone body
171	hcc	hemcylcon.f12	Cell heights at each body point in block 2 for the hemisphere-cylinder-cone case
172	hcc	hemcylcon.f18	Indices defining the grid surfaces to be viewed when the hemisphere-cylinder-cone case is run in the graphical version
173	hcc	hemcylcon.f16	Control scalars for the re-starting the hemisphere-cylinder-cone case
174	hcc	hemcylcon.res.out	Output file (printout file) for the hemisphere-cylinder-cone case, after restart
175	box	boxtm.f10	Control scalars for the first variation on the box case -- using Thomas & Middlecoff clustering terms
176	box	boxtm.out	Output file (printout file) for the first variation on the box case -- using Thomas & Middlecoff clustering terms
177	box	boxopt.f10	Control scalars for the second variation on the box case -- using locally optimal relaxation parameter (Ω)
178	box	boxopt.out	Output file (printout file) for the second variation on the box case -- using locally optimal relaxation parameter (Ω)

Table 12. Data Files for Example Cases (Files 160 - 178)

THE BASIC BOX CASE

As said above, this case is a box with edges conforming to what would be a cube, but with sinusoidal bumps on all six sides. The number of cycles comprising the bumps varies between different directions and different faces. The amplitudes of the bumps are large (approx. $\pm$ 0.18 times the length of a side) in the centers of the faces, and reduced to near zero at the edges. The distribution of points on the faces is consistent with the clustering requested in the interior.

A look at boxall.f10 will reveal four sets of "iterations" lines. On them it can be seen that we are using the coarse/fine technique. This is made possible by the fact that the

numbers of points in each direction (see the "dimension" line), 46, 49, and 52, are all of the form 3n+1 (for different values of integer n). This requires prior planning.

The "iterations" lines also reveal the basic recommended "game plan" for running the code:

- a few coarse iterations with control turned off to smooth out the sometimes odd initial conditions, followed by
- enough coarse iterations with control turned on to more or less converge the coarse case, followed by
- enough fine iterations to finally converge the case. This last step is first performed with control terms turned off (not shown) to verify the correctness of the boundary treatment by generating an uncontrolled grid, and then with control terms turned on (as shown) to achieve the desired control of cell size and skewness near boundaries.

A plot of the iteration history is seen in Figure 2a. Two functions are shown on this graph. The first is Maxmove, the amount that the point which moves the farthest between successive iterations moves. The other function is Avemove, the average amount that all the points move between successive iterations. It should be emphasized that these functions are plotted in the user's own units. The box in this example is nominally a cube one unit on a side, and in reference to that we see that the Maxmove is converged to about $0.2*10^{-6}$ units and Avemove is converged to about $0.25*10^{-7}$ units. Such a convergence history, with those functions starting high, doing some wiggling, and then being reduced monotonically by several orders of magnitude, is typical of a properly converged solution. If these two functions don't eventually decay by several orders of magnitude, then convergence has not been achieved, and the resulting grid will probably not conform to the user's requirements. Most users in most cases will not really need such a high degree of convergence as shown here, and so may consider reducing the number of iterations. The small wiggles at the very end of the Maxmove plot indicate that the solution has been converged to the limit imposed by round-off error. All the cases shown here were run on a Silicon Graphics Indigo2 R4400 workstation, a machine which does 32-bit floating-point arithmetic.

This case has also been run on a CRAY C-90, where the resulting convergence history is somewhat different but has the same general trends. It should be noted, however, that in the vectorized CRAY version the difference stencil used in the Point-SOR solver is different. At some of the points at which new data is used in the serial version, old data must be used in the vectorized CRAY version. This is necessary to avoid data dependency problems. Although vectorization on the CRAY greatly reduces the time per iteration, the modified difference stencil tends to increase the number of itertions required.

Figure 2b is a plot of two different functions, plotted against the same horizontal scale, for the same case. One is Pqrmax, the maximum of the absolute values of the S&S-type RHS terms over all boundary surfaces where control is activated. This function should, as seen here, rise to some large value and then level off. This leveling-off indicates that the S&S-type RHS terms, which are found iteratively, have been converged. The other function plotted here, Pqrcor, is the maximum of the absolute values of the change in (or correction of) the S&S-type RHS terms over all boundary surfaces where control is activated. In not-too-precise terms, Pqrcor could be thought of as the derivitive of Pqrmax. Thus Pqrcor tending toward zero also indicates that the S&S-type RHS terms have converged.

Figure 3a shows the left-hand boundary surface, with three more-or-less horizontal interior grid surfaces, in the finished grid. This is what grids look like when run in the graphics version of the code, except that here the Figure is reduced to grayscale while the acutal screen is in color. The body-conforming nature of the grid is shown, as is the successful imposition of a small and uniform cell height and near-orthogonality near he boundary surfaces. Elliptic methods, of which this is one, are known for generating very smooth grids, and that is seen here.

Figure 3b shows two intersecting interior grid surfaces in the same grid, at a similar orientation.

Another look at the file10 for this case will show that omega has been raised from its default value of 0.7 to a value of 1.0 for this case. The user may wish to experiment with raising that parameter further to achieve faster convergence. A value of 1.3 or 1.4 should converge, and it should cause the number of required iterations to be reduced by half.

Figure 2. Convergence Histories for Basic Box Case.

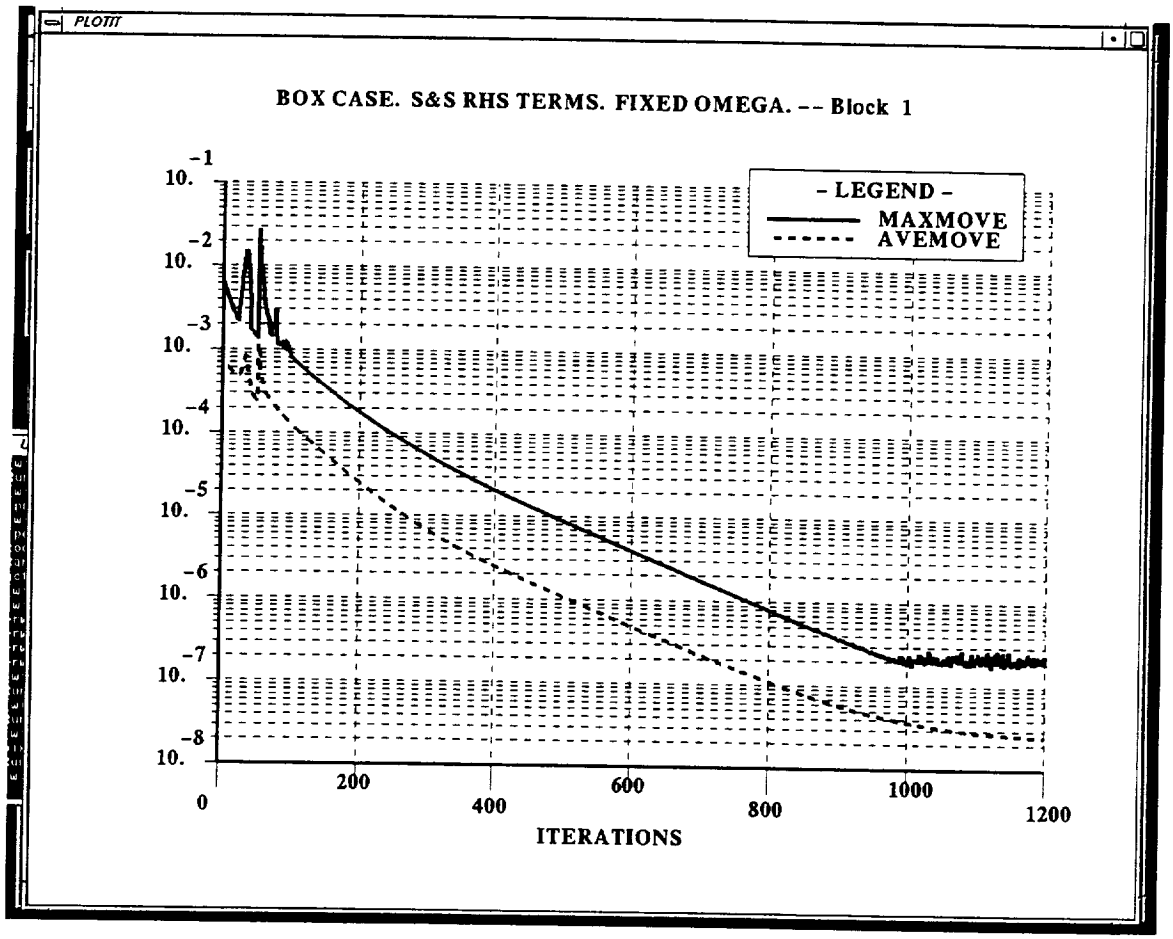

Figure 2a. Point Movement Functions.

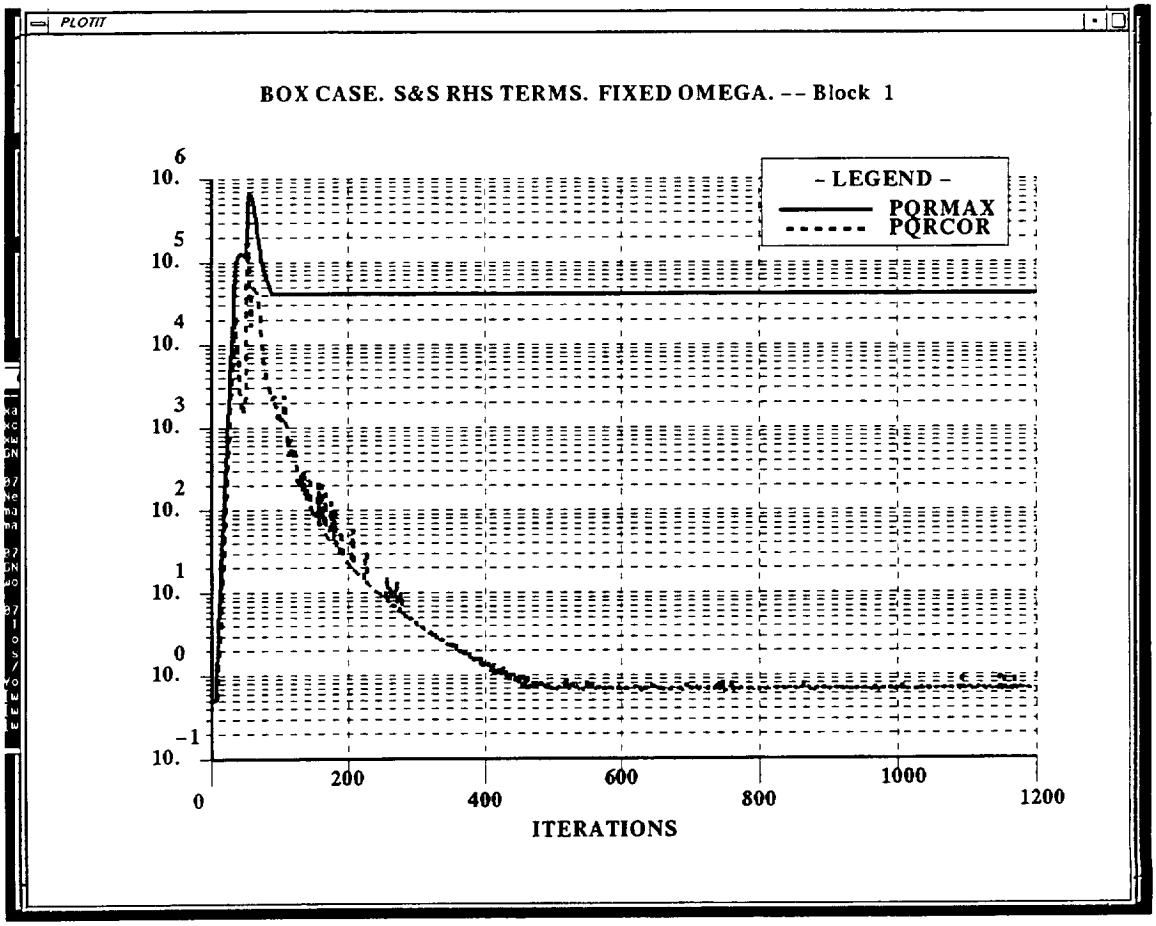

Figure 2b. Control Term Functions.

Figure 3. Wavey-Sided Box Grid.

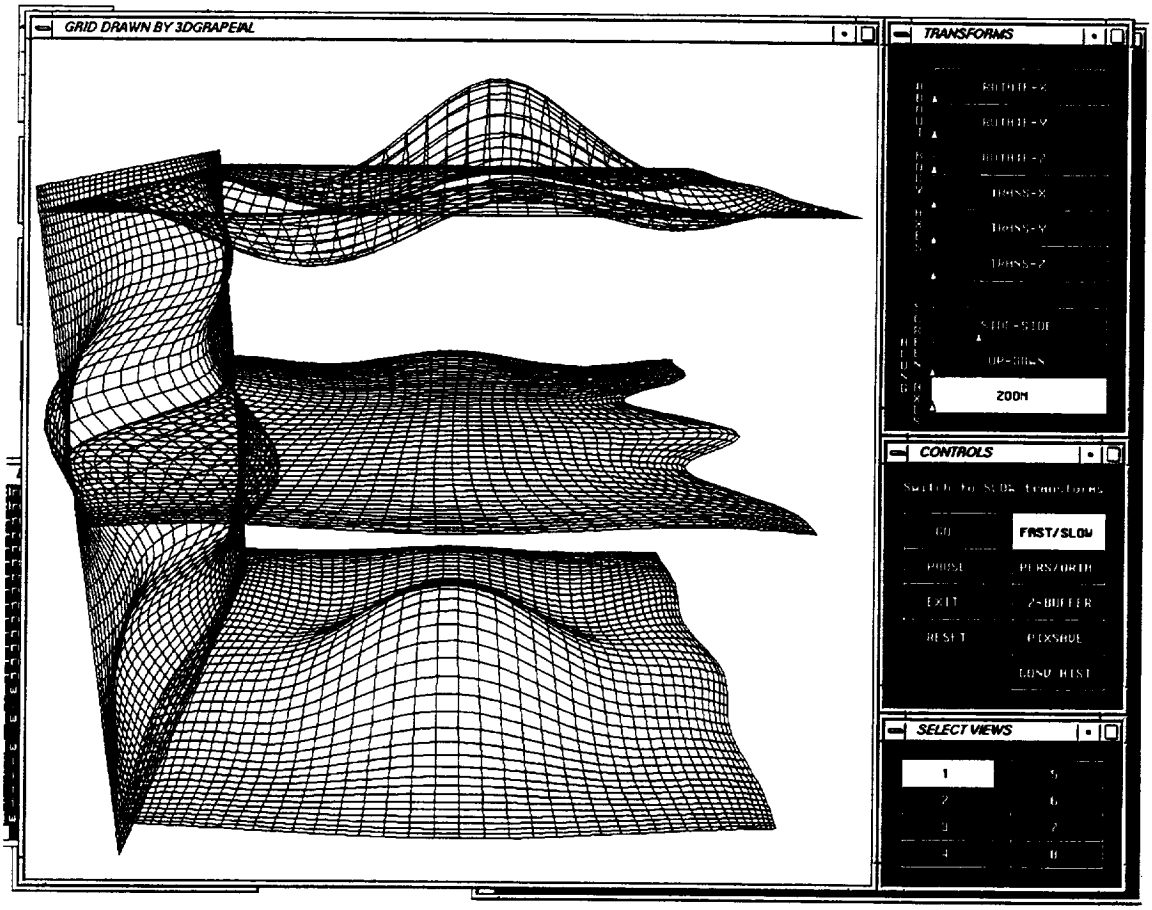

Figure 3a. Exterior Surface With Three Interior Surfaces. Graphical User Interface (GUI) Also Shown.

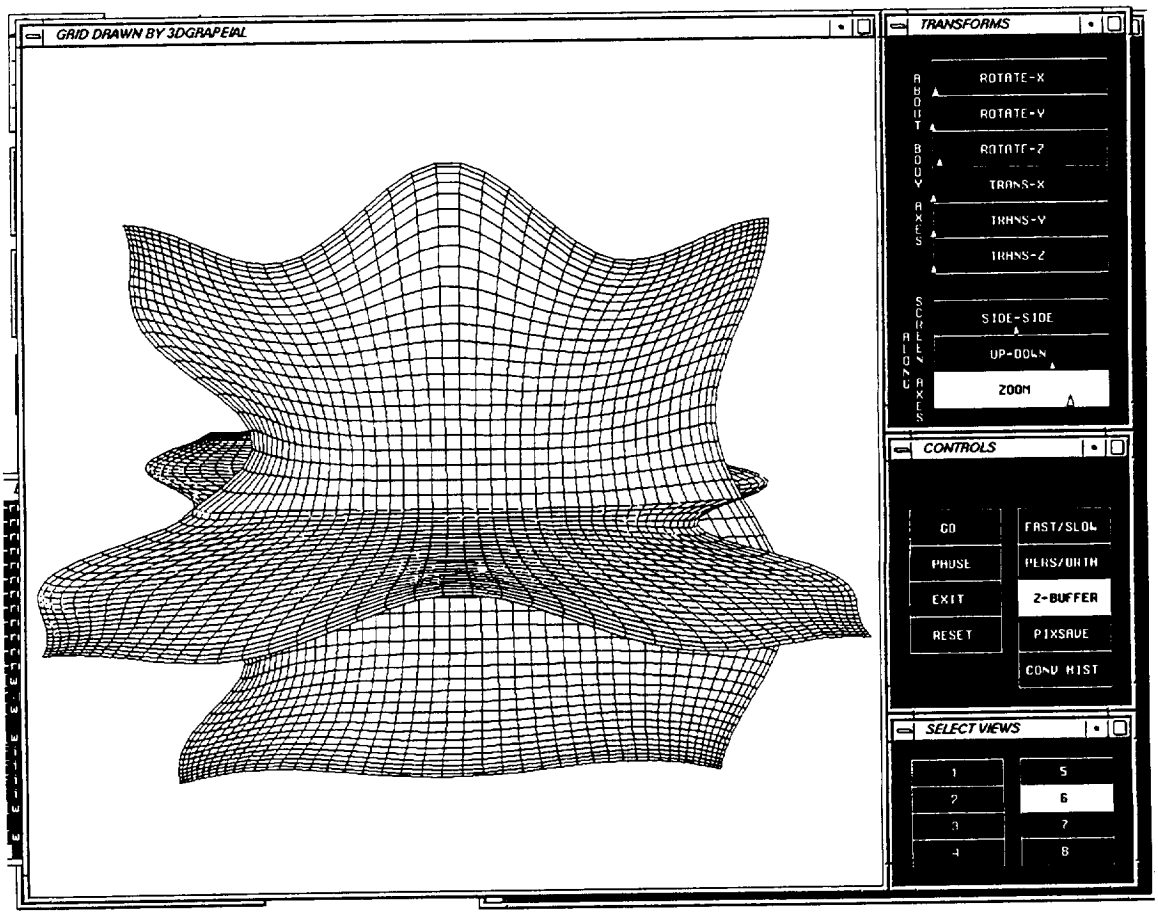

Figure 3b. Two Intersecting Interior Surfaces.

THE WING WITH FLAT PLATE EXTENSION CASE

One of the principal new features of the "Ames-Langley Technology Upgrade" is an improved ability to bend a grid across a sharp corner in a boundary surface. The ultimate example of that is wrapping a grid across the edge of a flat plate. The need to do that arises more often than one might imagine, with this case being an example. A C-type grid is wrapped about the leading edge of a wing, and then for topological purposes that wing is extended in the spanwise direction as a flat plate.

Figure 4a shows the wing, having a NACA 0012 section, its extension in the spanwise direction, and another planar extension in the wake region. All this constitutes the body boundary surface. It is shown from the root end of the wing looking toward the tip.

Figure 4b is from a different point of view -- from the tip end looking back toward the root. The flat plate extension in the spanwise direction is clearly shown, along with portions of two grid surfaces wrapping around the wing and its extension. An extreme close-up of an interior grid surface wrapping around the leading-edge of the flat plate extension is shown in Figure 4c.

The convergence history for this case is shown in Figures 5a and 5b.

Figure 4. Wing With Flat-Plate Extension Grid.

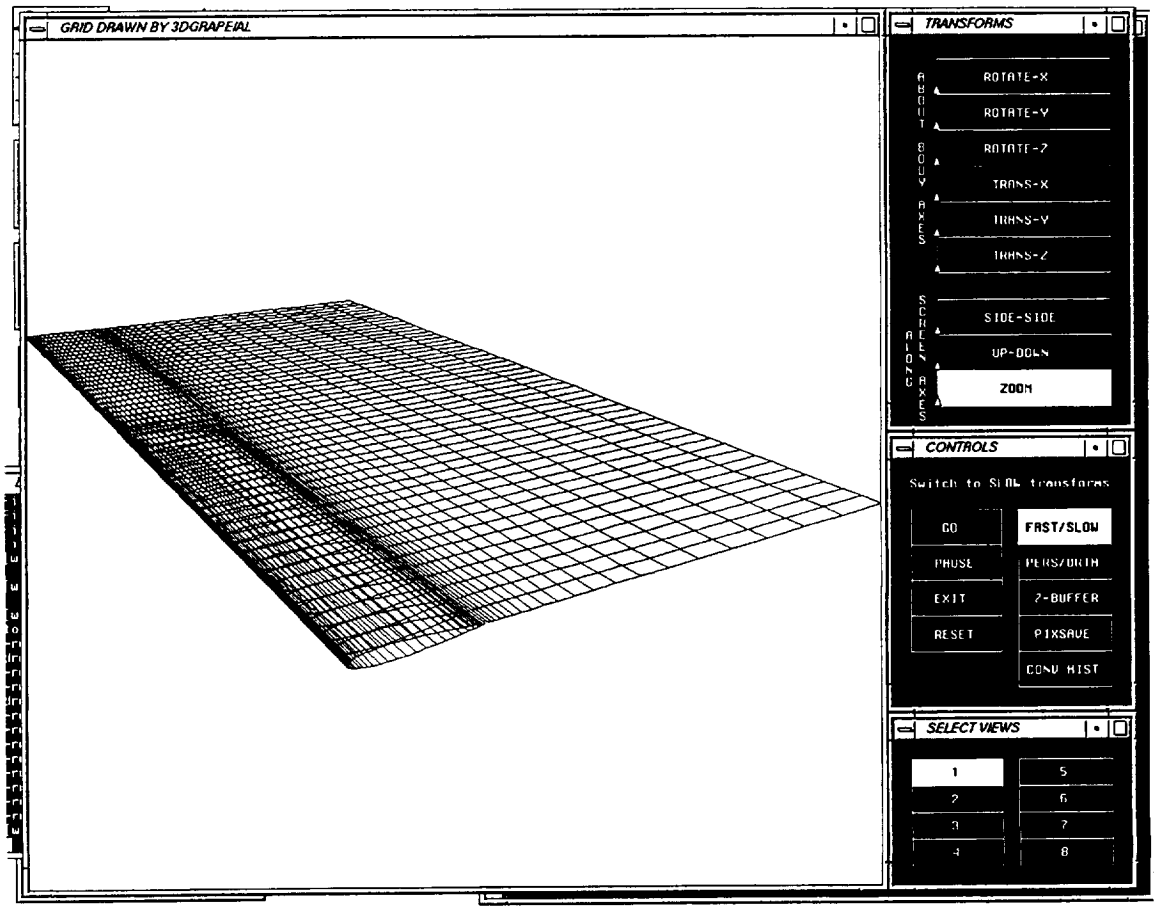

Figure 4a. Body, Consisting of Wing With Flat-Plate Extension in Spanwise Direction and Planar Extension in Streamwise Direction.

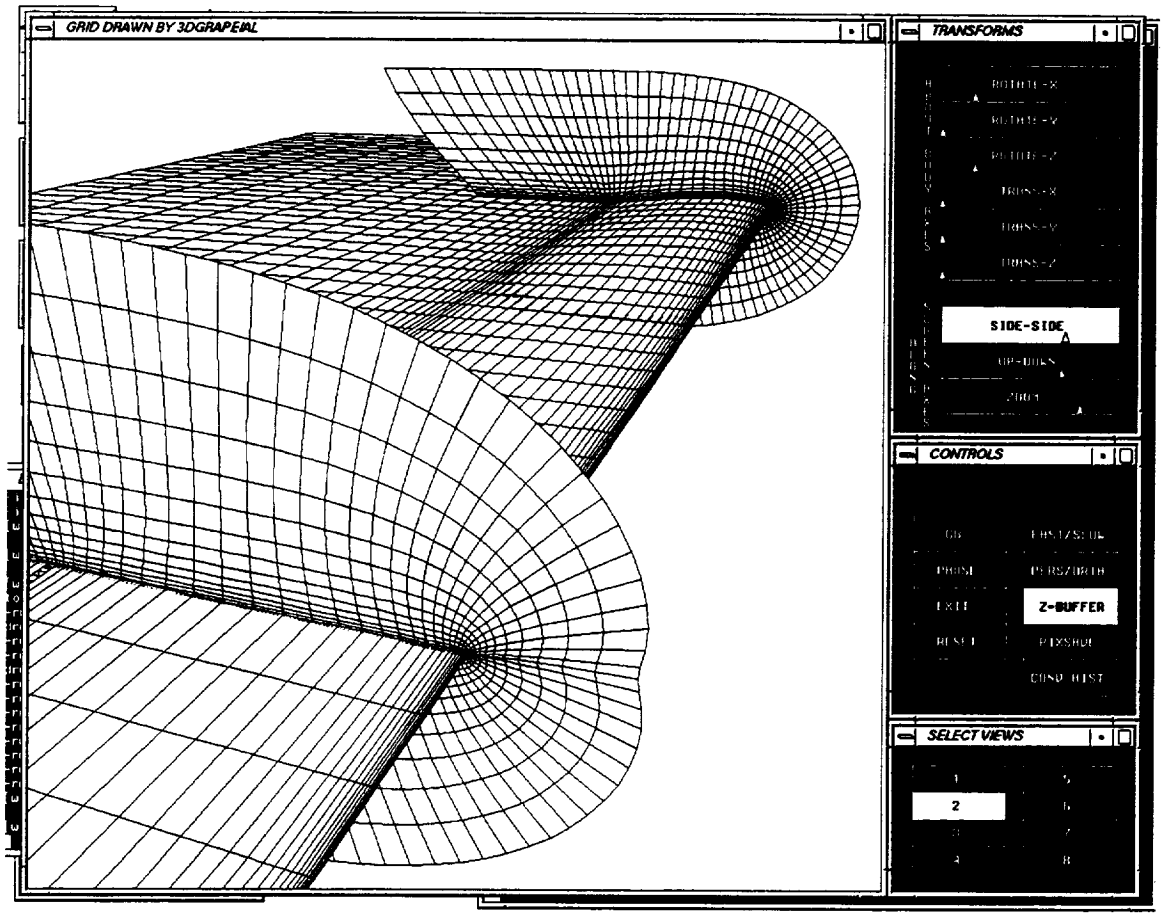

Figure 4b. Wing and Portions of Two Interior Surfaces Wrapping Around.

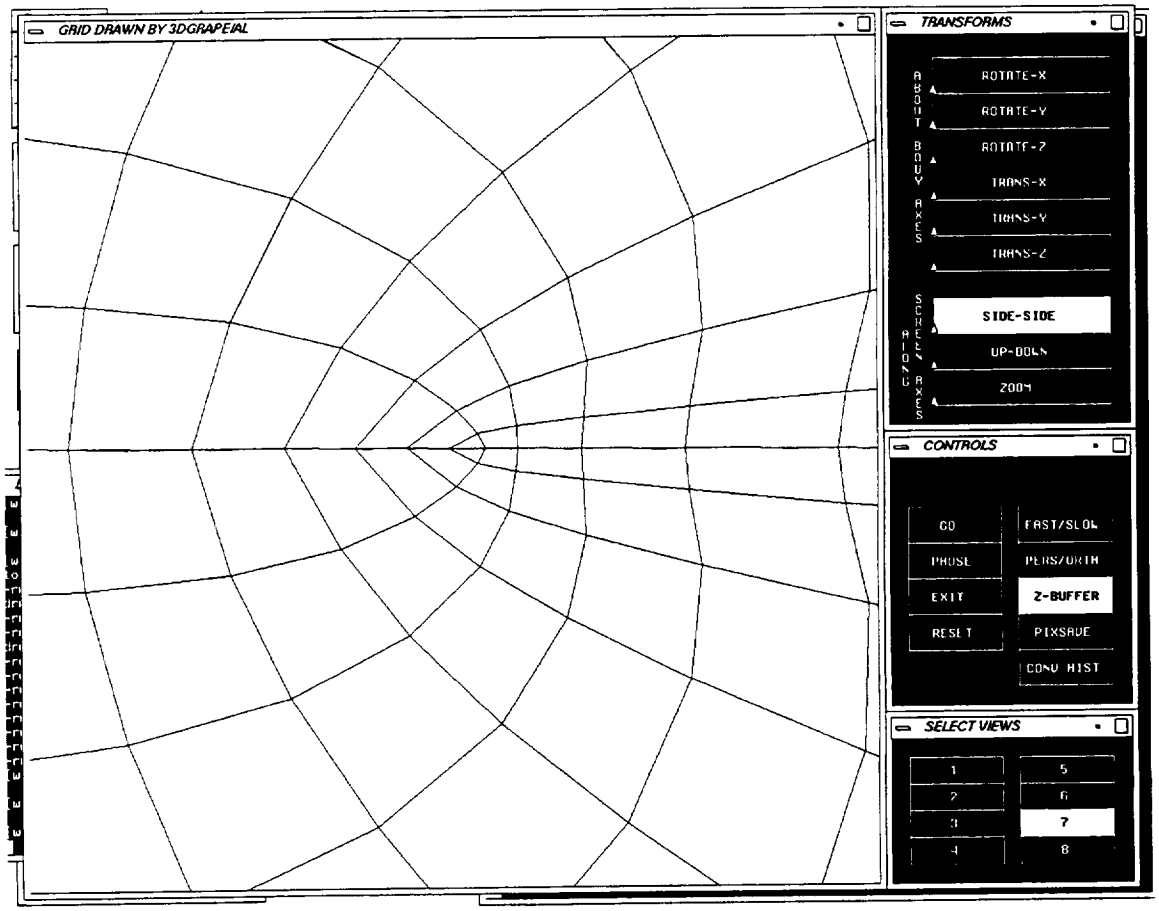

Figure 4c. Close-Up of Interior Surface Wrapping Around Leading-Edge of Flat-Plane Extension

Figure 5. Convergence Histories for Wing and Flat-Plate Extension Case.

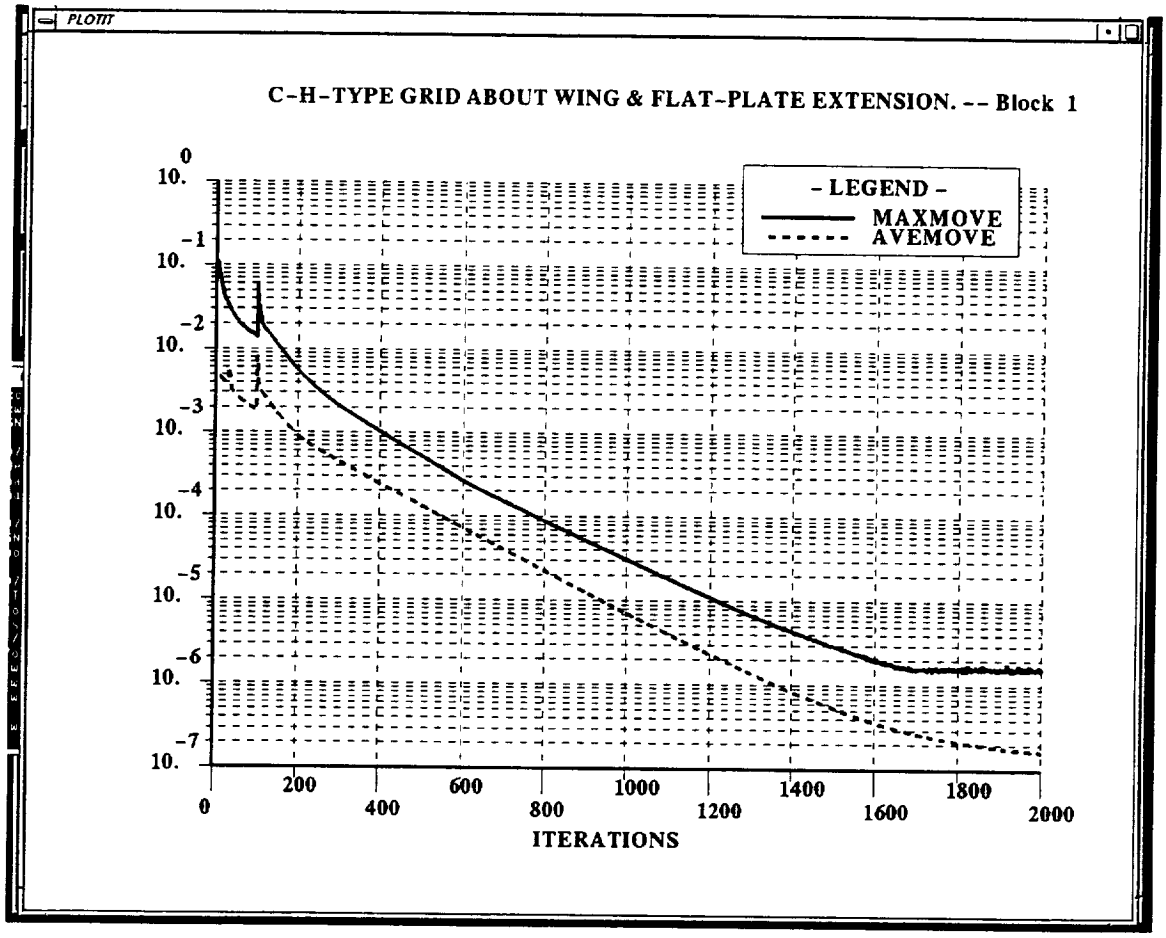

Figure 5a. Point Movement Functions.

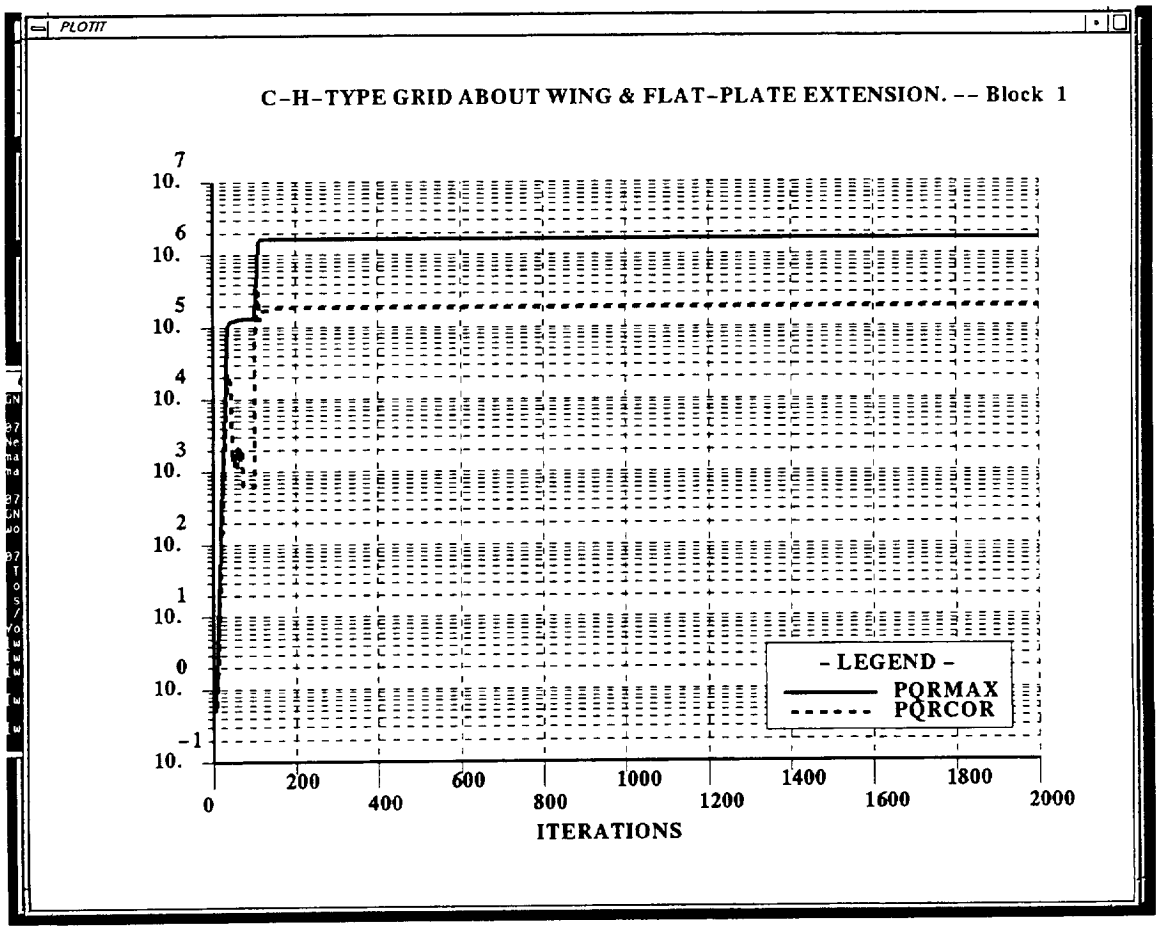

Figure 5b. Control Term Functions.

THE HEMISPHERE-CYLINDER-CONE CASE

The value of the hemisphere-cylinder-cone case is that it illustrates almost all possible boundary treatments. It is also a multiple-block case, and the freezeblock feature is used. We have also chosen to illustrate reading cell heights from file12.

The body is shown in Figure 6a, with an axis trailing out the back. Many grids, including most aircraft grids, have planar symmetry and so only one half of the configuration is actually used. That is the case here; if we define the hemisphere as the front (or nose) of the configuration, then the left half is missing and only the right half is gridded.

This case also illustrates the use of spherical topology in the block about the hemispherical nose. Spherical topology should be used whenever the grid has a spherical axis, such as is seen ahead of the nose in this case. If the use of spherical topology is called for by the actual problem being gridded, and spherical topology is not chosen in this code, decidedly odd behavior can result on and near the axis. Note that in this case it was found necessary to reverse the handedness of this block because of the use of spherical topology, as described in a previous section on the "handedness" input data line.

Some of the interior grid surfaces are shown in Figure 6b.

Returning to the file10 for this case, the "write-for-restart" line shows that a restart file was written and a restart was performed. The reason for this is two-fold. First, it simply illustrates the code's restart capability. Secondly, a small problem was found in block 3, about the axis behind the body. As the code continued to iterate points moved closer and closer to the X-axis, until their coordinates in the Y- and Z-directions approached round-off. This can cause problems, so we employed the simple artifice of ceasing to iterate on block 3 after 500 iterations (out of a total of 2,000). The input parameter "freezeblock=" is set to "ye" for block 3 in the restart input data file hemcylcon.f16.

The restart was performed after 500 iterations. Plots of convergence histories for blocks 1 and 2, shown in Figures 7a through 7d, show a small disruption at that point, from which the code quickly recovers. In other cases testers have seen no disruption of the convergence hisory at a restart. Numerical values for the convergence histories for blocks 1 and 2, in their entirety from start to finish, are shown on the supplied output file, hemcylcon.res.out, taken from the restart run. The convergence history for block 3, though shown in Figure 7e, appears in numeric form only on the output file resulting from the starting run which is not included on the program distribution tape. Note that there is no plot of the convergence of control terms for block 3, because all boundary surfaces in block 3 have control turned off.

Figure 6. Hemisphere-Cylinder-Cone Grid.

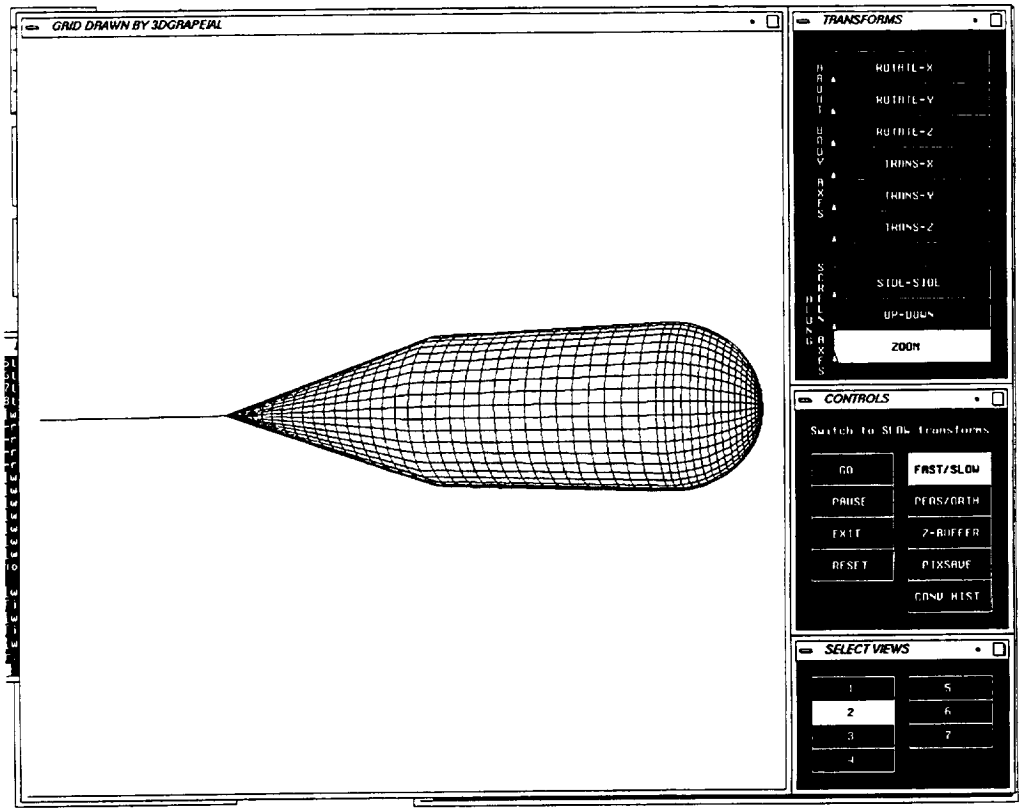

Figure 6a. Body With Trailing Axis.

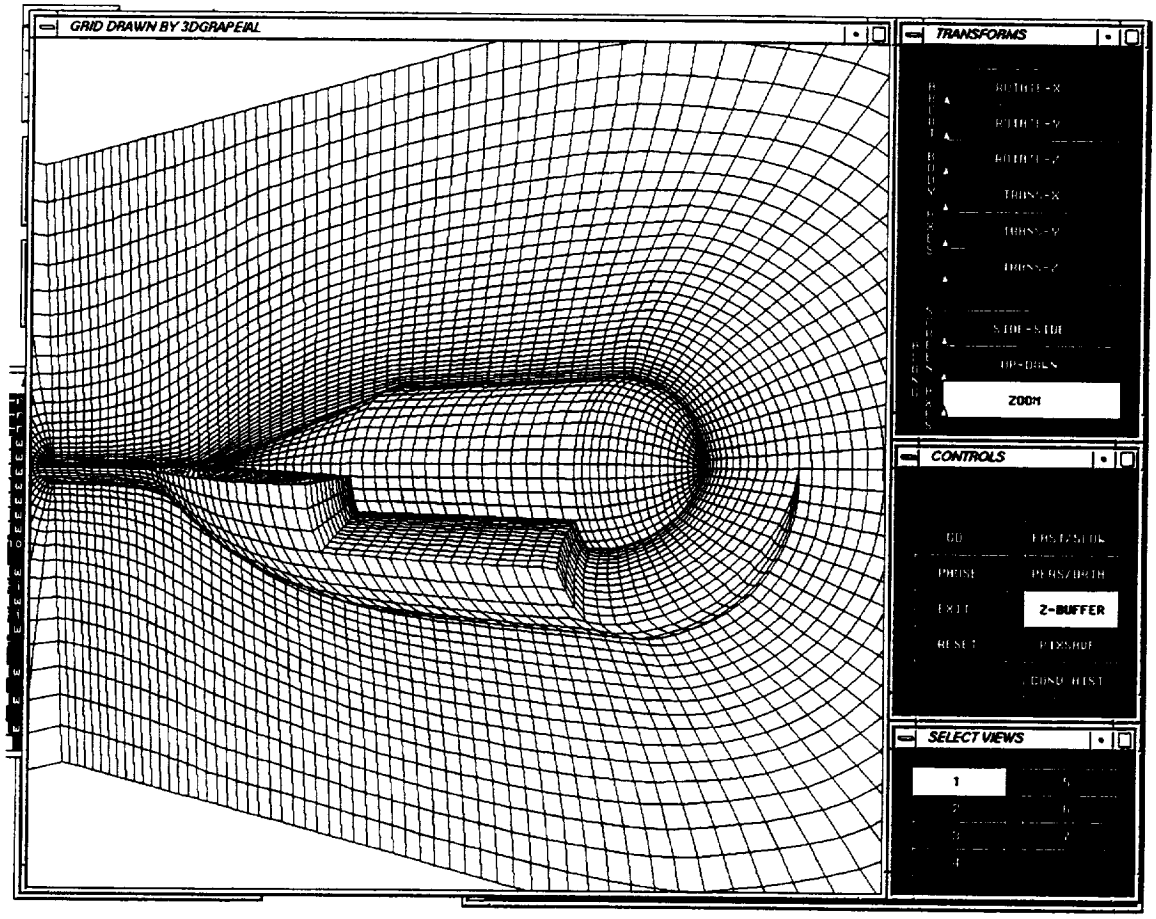

Figure 6b. Body, Symmetry Plane, Body, and Selected Interior Surfaces.

Figure 7. Convergence Histories for Hemisphere-Cylinder-Cone Case.

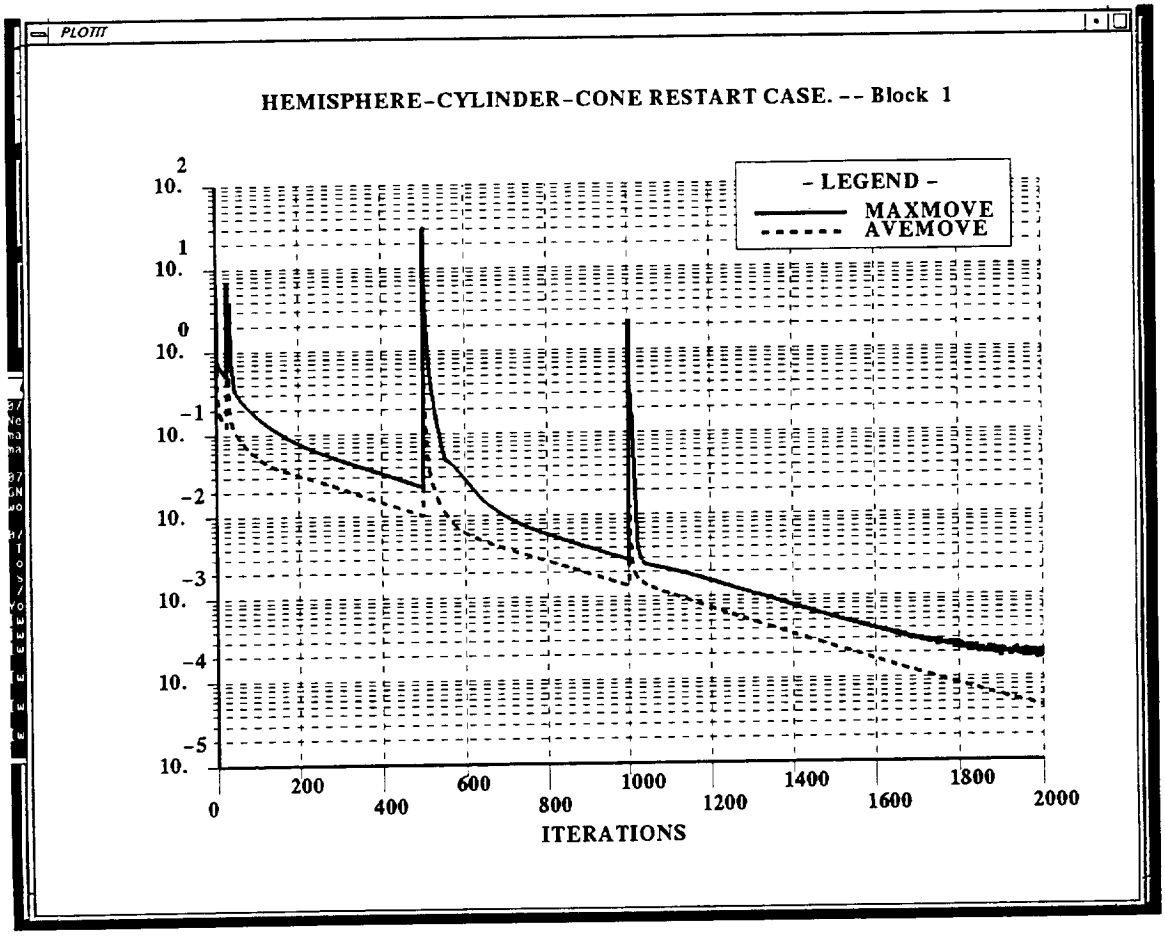

Figure 7a. Point Movement Functions for Block 1.

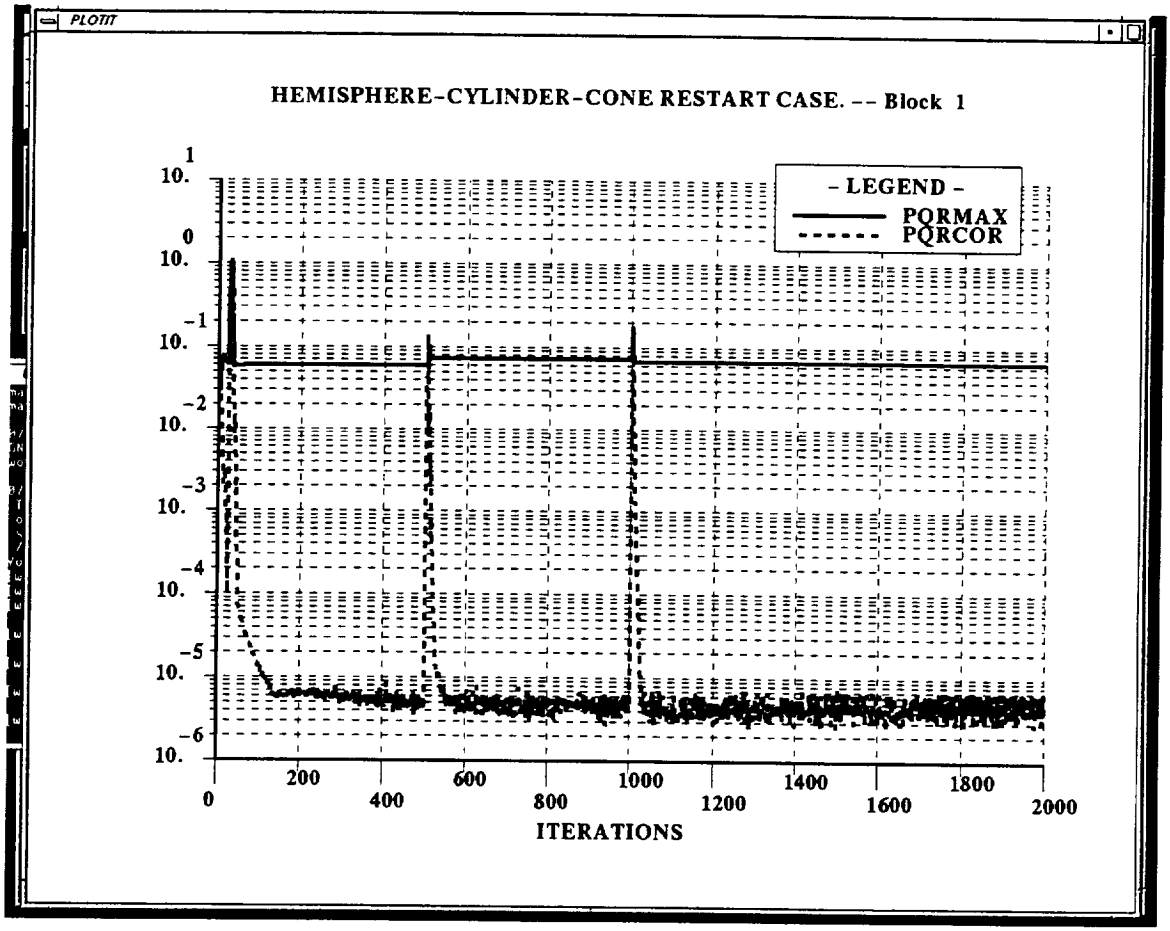

Figure 7b. Control Term Functions for Block 1.

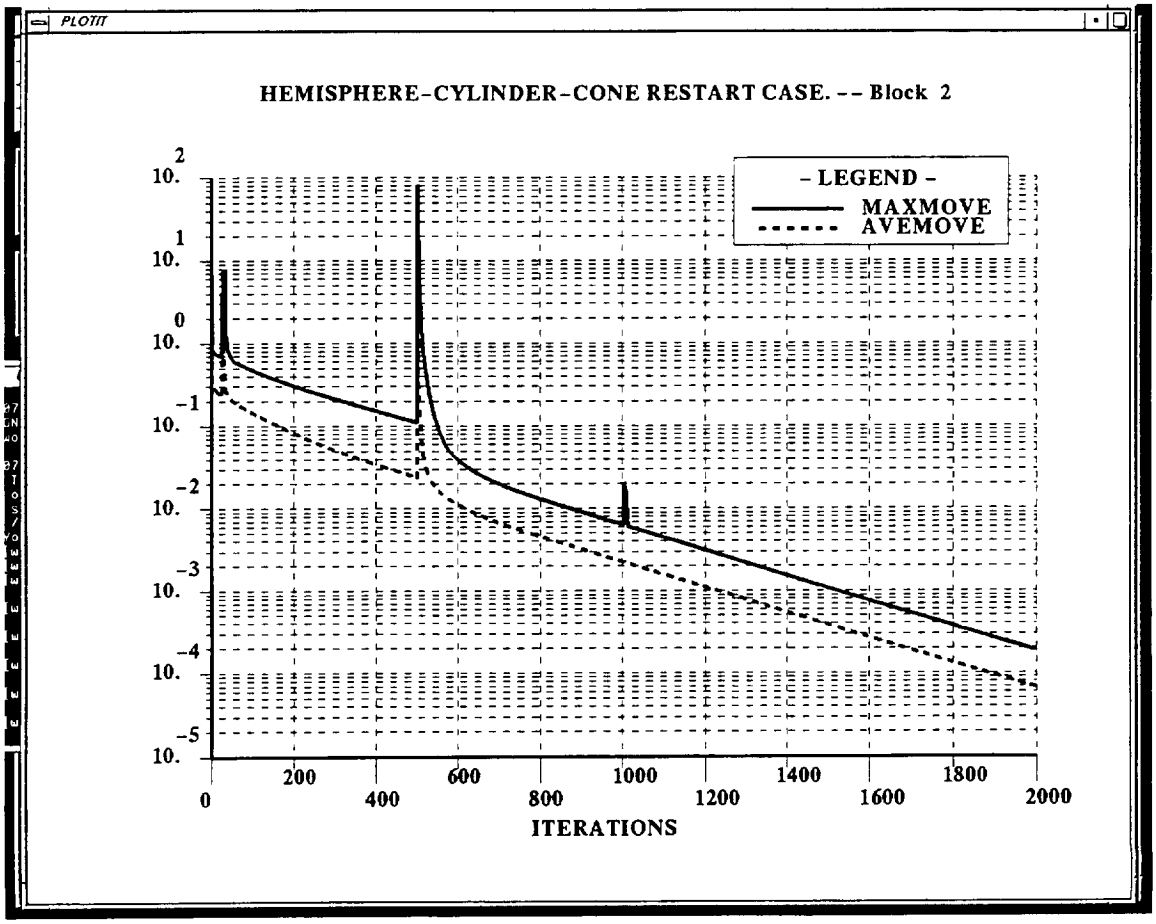

Figure 7c. Point Movement Functions for Block 2.

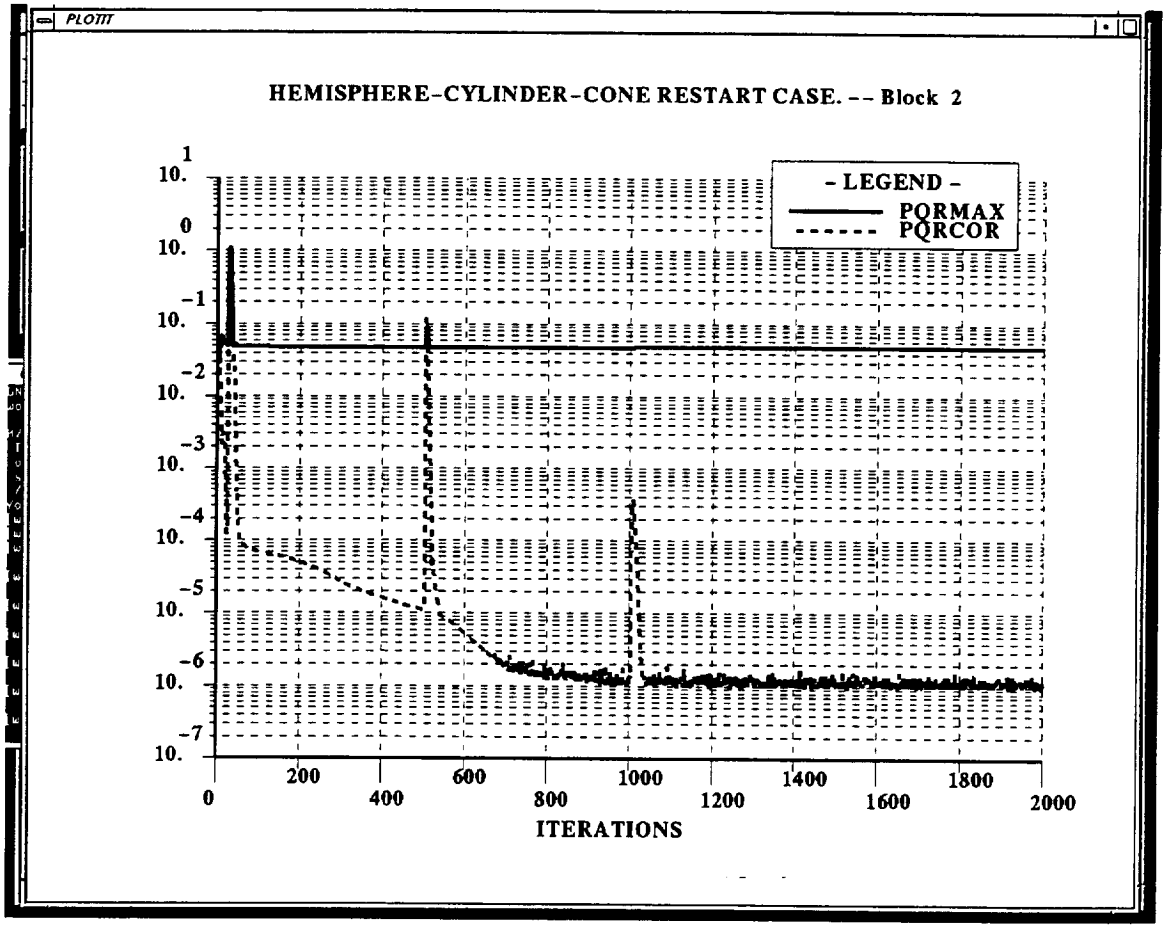

Figure 7d. Control Term Functions for Block 2.

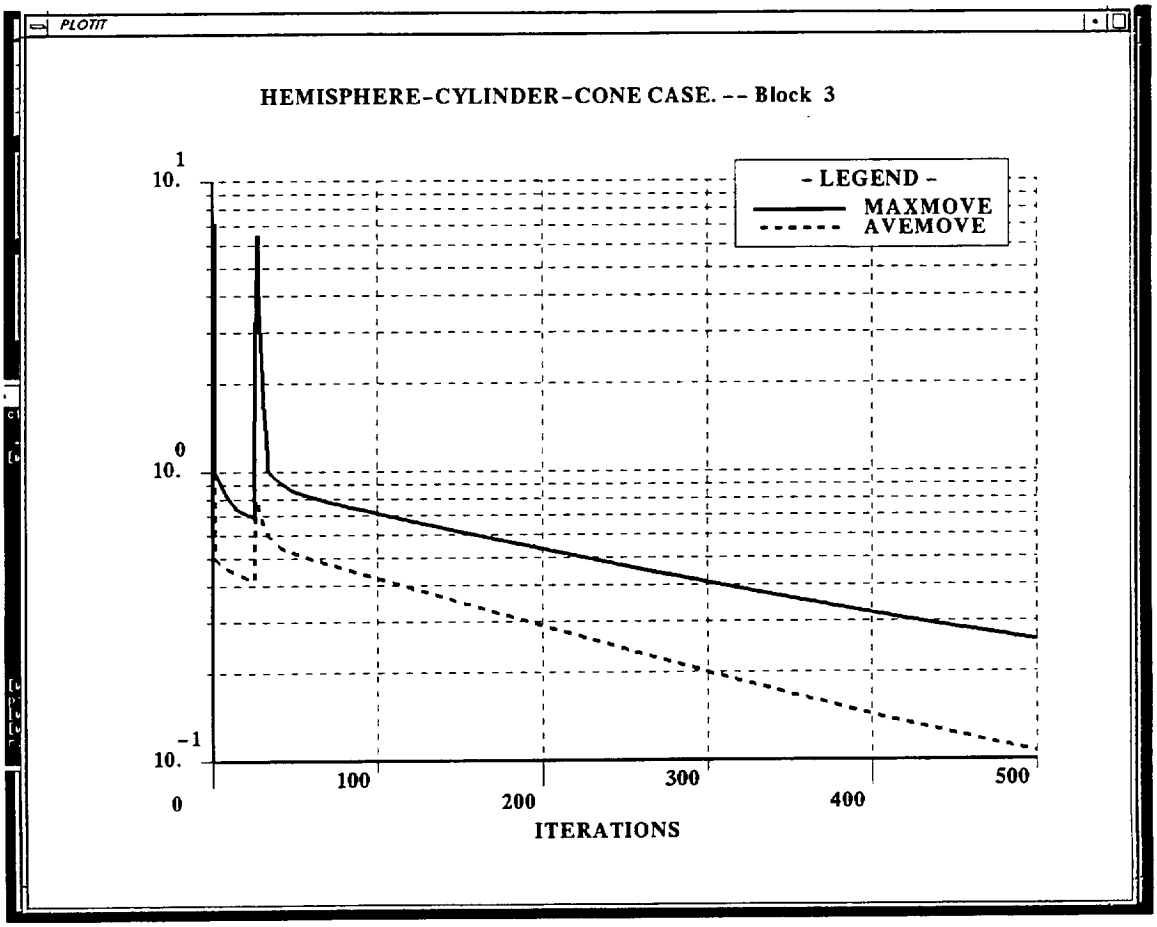

Figure 7e. Point Movement Functions for Block 3.

FIRST VARIATION ON THE BASIC BOX CASE -- THOMAS & MIDDLECOFF CLUSTERING TERMS

The basic box case used Steger and Sorenson clustering terms. This case uses both S&S terms and Thomas and Middlecoff terms, with blending between the two. See the "iterations" lines in boxtm.f10. The differences between the grids are difficult to see in plots such as appear in this manual, so no attempt is made to illustrate them. The interested user should generate his own results and look closely at them. For some cases grids generated this way are superior to those generated with S&S terms alone. Such things are highly case-dependent, so the user will have to experiment and come to his own conclusions.

The convergence history for this case is presented in Figures 8a and 8b. The S&S RHS terms for this case shown in Figure 8b converge in about 350 iterations, as opposed to 425 iterations for the basic box case shown in Figure 2b. The point movement functions shown in Figure 8a converge in about 650 iterations for this case, as opposed to about 1,000 iterations for the basic box case shown in Figure 2a.

Figure 8. Convergence Histories for Box Case With Thomas & Middlecoff RHS Terms.

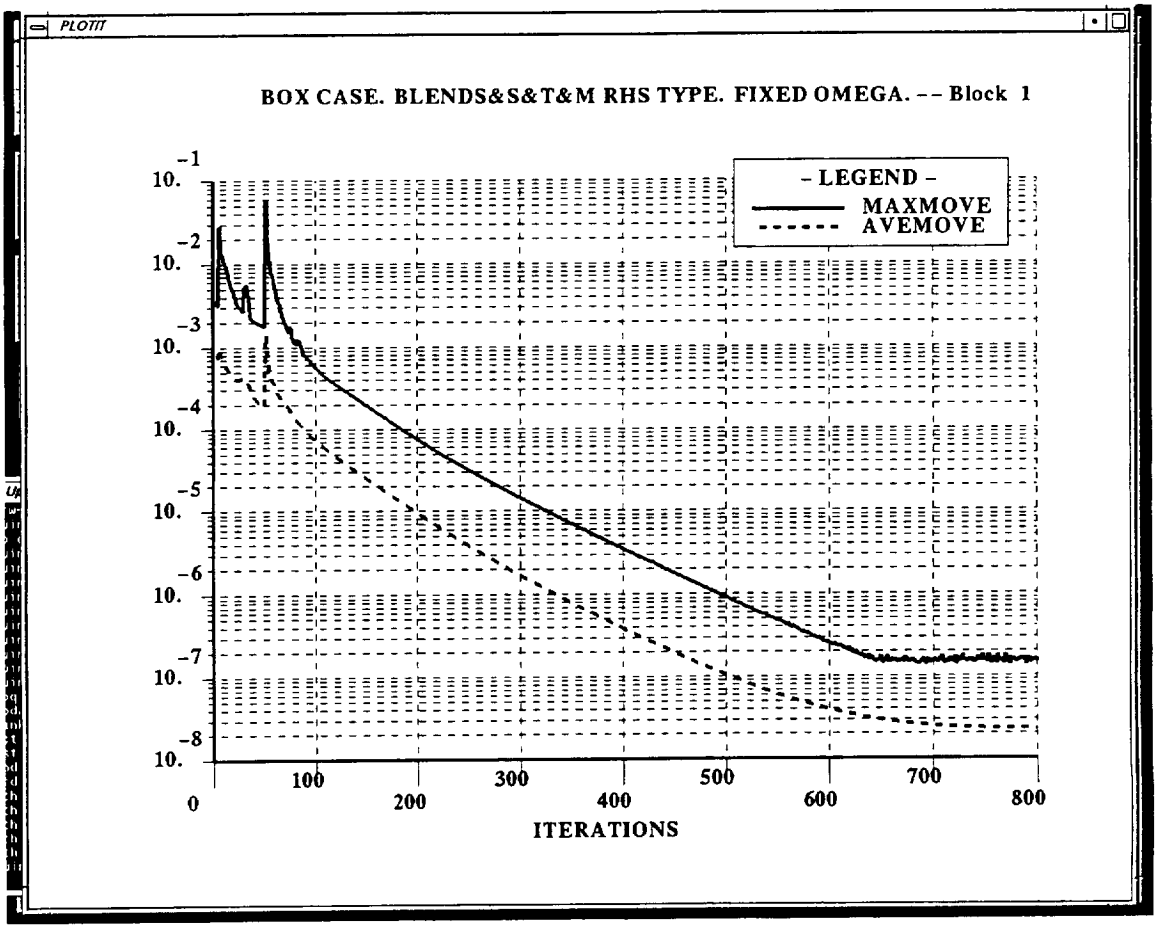

Figure 8a. Point Movement Functions.

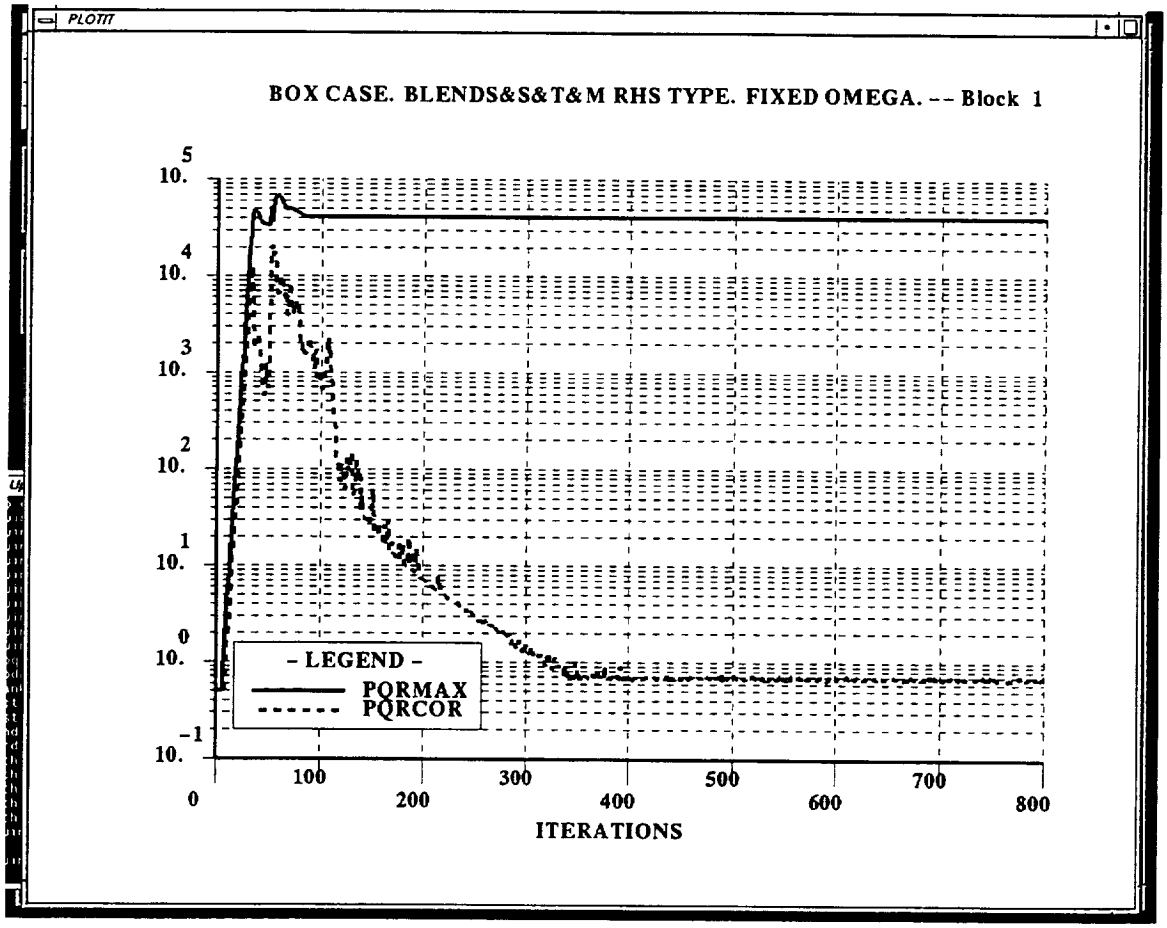

Figure 8b. Control Term Functions.

SECOND VARIATION ON THE BASIC BOX CASE -- LOCALLY OPTIMUM RELAXATION PARAMETER (Ω)

This second variation on the basic box case is one which uses locally optimum relaxation parameters (Ω). Note by inspection of file boxopt.f10 that it was found necessary to reduce "how-much" to 0.6 from its default of 0.7 to correct a failure to converge. But with that parameter set that way, it did converge faster than the basic case, as seen in Figures 9a and 9b. The S&S RHS terms for this case shown in Figure 9b converge in about 350 iterations, as opposed to 425 iterations for the basic box case shown in Figure 2b. The point movement functions shown in Figure 9a converge in about 800 iterations for this case, as opposed to about 1,000 iterations for the basic box case shown in Figure 2a.

Figure 9. Convergence Histories for Box Case With Optimum Omega.

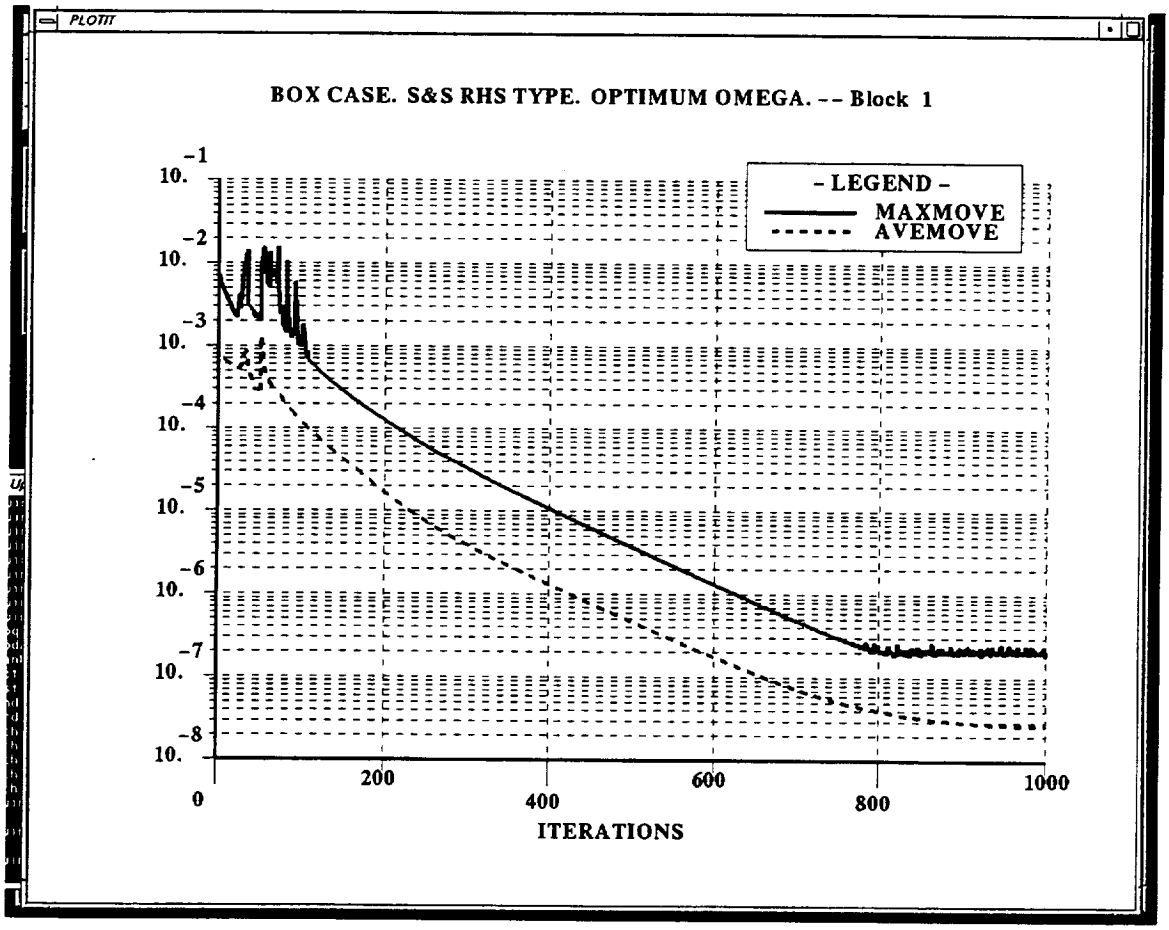

Figure 9a. Point Movement Functions.

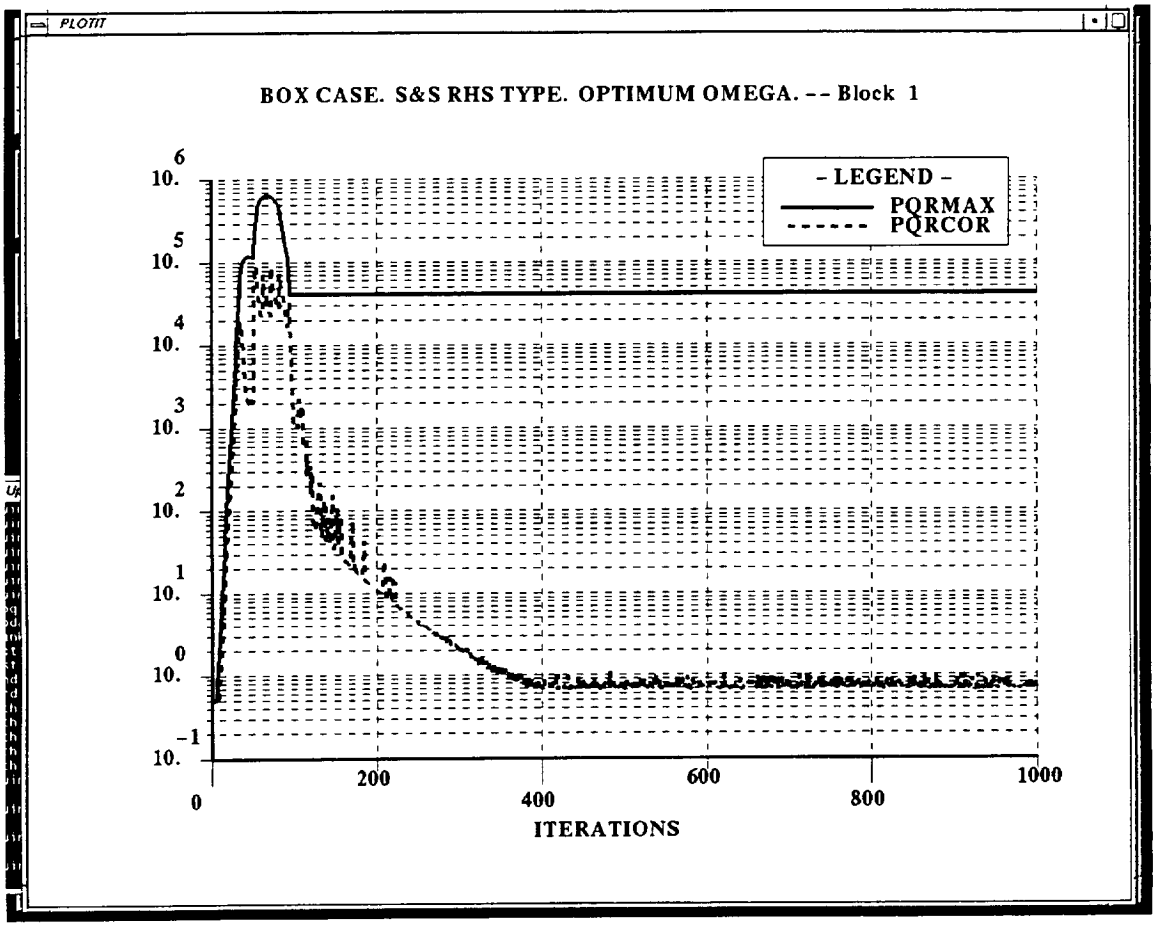

Figure 9b. Control Term Functions.

INPUT FILTERS

PREGRAPE/AL PROGRAM

PREGRAPE/AL is another program, supplied as a companion to 3DGRAPE/AL. Some users use the popular program GRIDGEN to do surface modeling and surface gridding. PREGRAPE/AL reads output from GRIDGEN (which could be input to GRIDGEN3D), and one other file, and outputs files which serve as input to 3DGRAPE/AL. Thus 3DGRAPE/AL can take the place of GRIDGEN3D. PREGRAPE/AL reads the *.bnda and *.mlga files which are output from GRIDGEN, and the *.ctrl file the user makes for this program. It outputs file10, file12, file16, and file18 for input to 3DGRAPE/AL.

Note: --> GRIDGEN, as used herein, refers to releases through no. 8 of that code. As of this writing there was an error in the .bnda file produced by the new version 9 of that code, which prevented both this code and GRIDGEN3D from running properly. When that error is fixed, this code will be made to run with GRIDGEN version 9.

Files which comprise PREGRAPE/AL are listed below:

File number:	Sub-directory:	File name:	Purpose or contents:
179	pre	pregrapeal_p.f	Main program of the PREGRAPE/AL program. It reads the *.bnda and *.mlga files which are output from GRIDGEN, and the *.ctrl file the user makes for this program. It outputs file10, file12, file16, and file18 for input to 3DGRAPE/AL.
180	pre	arlndist_p.f	Calculate the blending coefficients based on arclength rather than the actual parametric index of a given face

181 182 183 184 185 186 187 188 189 190 191	pre	blnkfl7_p.f blnkfl8_p.f blnkfl10_p.f blnkfl12_p.f blnkfl14_p.f blnkfl15_p.f blnkfl40_p.f blnkfl60_p.f blnkfl76_p.f blnkfl79_p.f blnkfl80_p.f	Copy non-blank characters from an input string to an output string. The length of the string is the number in the filename.
192	pre	dsout3_p.f	Utilizes the LARCS or TFI interpolation to specify a delta S per point.
193	pre	ext2out_p.f	Convert the .job file to a .out file.
194 195	pre	getdfalt_p.f gtdfalt2_p.f	These subroutines are designed to extract the default file name from the user inputted control file name.
196	pre	maxdim_p.f	Determine the maximum value of the two indices in a face for the respective directions.
197	pre	newfile_p.f	Determine if the user wants to try entering another file or just stop.
198	pre	nofile_p.f	Tell the user if the file just opened exists.
199	pre	ntrp_p.f	Compute the angle to be specified on a face, based on interpolation from the edges.
200	pre	polchk_p.f	Determine whether or not a given face is a pole boundary.
201	pre	split_p.f	Uncompress the value n into two numbers ndiv,nmod based on m.
202	pre	tmflwcon_p.f	Determine if the face that has the boundary condition (ibc) should have orthogonality controlled or not.
203	pre	makefile.pre	Makefile to compile and link PREGRAPE/AL
204	pre	exf.ctrl	Control scalars to run PREGRAPE/AL
205	pre	exf.bnda	Output file from GRIDGEN
206	pre	exf.mlga	Output file from GRIDGEN

Table 13. PREGRAPE/AL Program and Data Files (Files 179 - 206)

In order to use the 3DGRAPE/AL code, just as the 3DGRAPE code, a definition of the configuration's surface is needed. To make 3DGRAPE/AL a complete system, a volume grid block and 2D parametric block-face grid generator are required. To provide this

information, PREGRAPE/AL, an interface code, was created to link GRIDGEN2D to 3DGRAPE/AL.

The PREGRAPE/AL code uses its own input file and GRIDGEN3D input data to generate the 3DGRAPE/AL control decks. Other data required to run 3DGRAPE/AL are provided by PREGRAPE/AL, including:

- UNIX shell scripts for 3DGRAPE/AL (and 3DVOLCHK, another program not included in this package).
- Generation of 3DGRAPE/AL control decks (files 10, 11 and 16).
- Cell heights for Steger and Sorenson forcing function controls using either 2DTFI or LARCS (file 12).
- Specification of incidence angle at a boundary (file 12).
- Parameter file generation for dimensioning 3DGRAPE/AL (params.h).
- UNIX script to compile and link 3DGRAPE/AL

The PREGRAPE/AL input file has the following form:

```
Working directory of 3DGRAPE runs     (a):/scr/salter/wood/sc1/
UNIX Script for CRAY,IRIS,ONYX,SUN    (a):iris
FLAGS ctd,face,dsi,3dj,3dg          (5i2): 0 0 0 0 1
Configuration name                    (a):Straight Cone #1 for UPS Study
Default basename                      (a):sc1
Block Information file (*.bnda)       (a):sc1.bnda
Face Information file (*.mlga)        (a):sc1.mlga
Interactive Visualization             (a):elv10-vol.view
#of Newstart iteration sequences     (i2):01
  Number of     Laplace(0)  Coarse/  Steger &    Thomas &    Relaxation
  Iterations    Poisson(1)   Fine    Sorenson   Middlecoff     Rate
    800           1           0         1          0          0.5
#of Restart iteration sequences      (i2):01
  Number of     Laplace(0)  Coarse/  Steger &    Thomas &    Relaxation
  Iterations    Poisson(1)   Fine    Sorenson   Middlecoff     Rate
    800           1           1         1          1          -.7
Sorenson init (0); 3DTFI (1)      (f12.6): 1.
Decay rates for each block/face   (f12.6):-40.
    Block   Face      Decay Rate
    Number  Number      Factor
      1       1         -1.00
      1       2          0.20
      1       3          0.35
      1       4          0.35
      1       5          0.30
      1       6          0.35
Orthogonality Control                 (i4):  -6
   Block   Face    Interp.   Interp.  Blending  Normalized    2DTFI  Incidence
  Number  Number  indx1->3  indx2->4  Function  Arc Lengths   LARCS    Angle
     1       1        2         2         3         1          2        1
     1       2        2         2         3         1          2        1
     1       3        1         1         1         1          2        0
     1       4        1         1         1         1          1        1
     1       5        1         1         3         1          2        1
     1       6        2         2         3         1          2        1
```

The file is read with formatted Fortran statements for those lines containing the colons ":" and the rest of the information is read with free formats. Two header lines are used for

understanding the input file for the iteration control sequences, orthogonality decay rates and the calculation type for determining cell size. These header lines are expected and will be read as 80 column character strings. The description of each line is tabulated below:

Line no.:	Format:	Description:
1,2	(a)	Header for the file.
3	(41x,a)	Directory to find all data, including the source code. Note: If the directory has a ~ in front of it, the script written will be for a C-Shell, as opposed to the default Bourne Shell.
4	(41x,a)	Type of machine 3DGRAPE/AL will use.
5	(41x,5i2)	Control flags for the types of data to be produced: Flag # Description: 1 Control deck generation. 2 File 11 construction for ``read-in-fixed'' data. 3 Cell size and incidence angle calculations. 4 3DGRAPE/AL UNIX script generation. 5 3DGRAPE/AL parameter file dimensioned based on grid computational limits.
6	(41x,a)	First comment line in the 3DGRAPE/AL control deck, typically used to label the control file for clarity.
7	(41x,a)	Default basename of GRIDGEN and 3DGRAPE/AL files.
8	(41x,a)	Truncated GRIDBLOCK ascii file name.
9	(41x,a)	GRIDGEN2D block face grid definitions.
10	(41x,i2)	Number of iteration sequences to be run in the ``newstart'' control deck.
10a-b	(a)	Header for columns of following data.
10c-?	(*)	Number of iterations, activation of orthogonality controls, coarse or fine solution, activation of the Steger & Sorenson source terms, activation Thomas and Middlecoff source terms and the relaxation rate for a specific set of iterations in the newstart control decks. The four middle colums require a (1-$>$YES/0-$>$NO), while the last column is a positive or negative number for the relaxation rate of grid point movement. A negative number is that percentage of the optimum value. A positive number is a constant to be used.
11	(41x,i2)	Number of iteration sequences to be run in the ``restart'' control deck.
11a-b	(a)	Header for columns of following data.

11c-?	(*)	Number of iterations, activation of orthogonality controls, coarse or fine solution, activation of the Steger & Sorenson source terms, activation Thomas and Middlecoff source terms andthe relaxation rate for a specific set of iterations in the restart control decks. The four middle colums require a (1-$>$YES/0-$>$NO), while the last column is a positive or negative number for the relaxation rate of grid point movement. A negative number is that percentage of the optimum value. A positive number is a constant to be used.
12	(41x,f12.6)	Volume grid initialization through Sorenson's method (#1), or optimized 3DTFI (#2).
13	(41x,f12.6)	Decay rate specification for the forcing functions. A negative number denotes the default. The default in the ``newstart'' deck is ``keep-default''. The default in the ``restart'' deck is the conversion from GRIDGEN to 3DGRAPE/AL using the absolute value of the decay rate read as the value of the EXPO variable, illustrated later. Otherwise a positive number is the number of block/face combinations that will use lines 12a-?.
13a-b	(a)	First line header for columns of following data.
13c-?	(*)	Grid block number, face number and decay rate to be used to exponentially decay the orthogonality controls into the volume. A negative number for the decay rate denotes ``keep-default''for the specified block/face combination in the newstart and restart. A positive value sets the ``keep-default'' in the newstart and the specified value in the restart.
14	(41x,f12.6)	Cell height control of each block/face combination and the angle of incidence of the grid lines to the boundary. If the number is negative, the TEAM nomenclature and options within GRIDGEN, are used. In this case, only pole boundaries and matching faces have no control. All other face types will have orthogonality. If this number is positive, it represents the number of block/face combinations with controls to be specified in the following format
14a	(a)	First line header for columns of following data.
14b	(a)	Second line header for columns of following data.
14c-?	(*)	Block number, face number, type of interpolations for adjoining opposing faces: Linear (1), Elliptic (2), Hyperbolic blending of both opposing pairs, use of normalized arc lengths for interpolations, use of LARCS or 2DTFI for iterpolations and the incidence angles to be used for each block/face combination.

Table 14. Description of PREGRAPE/AL input file

There are two basic methods for the computations of cell size and incidence angle used for each face with orthogonality activated. The first is 2DTFI, which only uses the 6th field in this portion of the input deck. The 6th field determines if the 2DTFI is done based on computational coordinates or the physical coordinates. To use the computational coordinates, place a 0 in the 6th field, other wise set it to 1 to use normalized arc-lengths based on the physical coordinates.

The second method of interpolation is to use LARCS or Local ARc-length Cell Sizing. Briefly, the LARCS method take blendings of the two opposing paired faces that connect to the specified face and combine the effects of each into a single blended surface representing a value. In the case of the cell sizes, the blended surface represents the cell sizes at each point on the surface, in computational or physical space. For example, if the cell sizes of face 5 are to be computed using the LARCS method, the two opposing paired faces that connect to face 5 are faces 1 & 2 and faces 3 & 4, respectively. First, each cell size distribution, dependent on the mating or connecting line to the specified face, are blended using bi-linear or elliptic interpolation, for each opposing paired faces. Then a hyperbolic blending function is used to blend the two interpolated surfaces into a single blended surface. Though the method may seem extensive, it produces much smoother distributions of cell sizes or incidence angles than 2DTFI.

The LARCS method requires the information in fields 3 through 6. Field 3 determines the type of interpolation between the first set of opposing faces (bi-linear(1) or elliptic(2)). Field 4 determines the type of interpolation between the second set of opposing faces. Field 5 determines which interpolated or blended surface of LARCS to use for specifying the cell sizes and incidence angles. The options are:

(1) Use the interpolated surface from the first opposing pair of faces.

(2) Use the interpolated surface from the second opposing pair of faces.

(3) Use the hyperbolically blended surface of both paired opposing faces.

For the opposing pairs of each face, the following table can be used:

Face:	Pair #1	Pair #2
1	5 & 6	3 & 4
2	5 & 6	3 & 4
3	5 & 6	1 & 2
4	5 & 6	1 & 2
5	1 & 2	3 & 4
6	1 & 2	3 & 4

Table 15. Opposing Face Pairs

Finally, the incidence angle specification provides an alternative to orthogonality. Utilizing the incidence angle specification, forces PREGRAPE/AL to compute the angle of incidence of each grid line emanating from a specified boundary, by interpolating the angles from the edges of that specified boundary.

NOTE: The computed cell heights/sizes and the boundaries with interpolated angles of incidence will have a different file name, but all required data will appear in file 12. The individual file names will have the following convention:

dsiblkBBfF.tcp -> Cell Sizes in TECPLOT data format

ntrpblkBBfF.tcp -> Incidence angles in TECPLOT data format

where BB represents the block number, and F represents the face number.

For the PREGRAPE/AL input, the user must specify a negative EXPO value to use the GRIDGEN3D to 3DGRAPE/AL conversion. For GRIDGEN3D, the default EXPO is 6.

Although the file seems involved and an added step, the PREGRAPE/AL code provides a lot of flexibility in the grid-generation process. The transition between GRIDGEN and 3DGRAPE/AL is smooth and efficient. This transition also enables the user to generate large grids easily and efficiently.

The method of volume grid generation typically used in conjunction with the 3DGRAPE/AL code is the following:

- Construct or obtain the surface of a configuration.
- Load the surface geometry definition into GRIDBLOCK of the GRIDGEN code.
- Construct the grid-blocking structure to be used, as well as setting CFD boundary conditions and face-matching definitions.
- Load the GRIDBLOCK output into GRIDGEN2D and create all defining faces of the grid-block structure. Note, there are six faces for each block.
- Output the face grid distributions (also referred to face definitions) and the boundary conditions to load into the GRIDGEN3D code.
- Set up the input file for PREGRAPE/AL and run it with the input [file].bnda and [file].mlga files usually read by GRIDGEN3D.
- Compile, link and execute the 3DGRAPE/AL code for the geometry to convergence or until the grid structure meets the needs of the user.
- Execute the volume checking portion of the 3DGRAPE/AL code to evaluate grid quality, and to determine if further iterations with the 3DGRAPE/AL code is necessary.

  The user may have to repeat the last 6 steps of the above method to obtain good grid distributions, or better parametric dimensional limits.

Warning --> When PREGRAPE/AL runs it produces a new params.h file. Thus, if it is run in the same directory as are stored the source files for the 3DGRAPE/AL code, it will wipe out the original params.h file copied from the distribution tape. Users should not run PREGRAPE/AL in the same directory as the 3DGRAPE/AL source files unless they are sure that they wish to employ this feature.

F10FILTER PROGRAM

The F10FILTER program is another program supplied with 3DGRAPE/AL which converts file10 input files as used in the earlier 3DGRAPE program into file10 input files which can be read by the new 3DGRAPE/AL program. It is a simple, straightforward Fortran program. The user should compile and link it in the obvious way, as with any other simple Fortran program, as:

```
f77 -o f10filter f10filter.f
```

To use it simply type the program name, followed by two file names -- the name of the old existing file10 data file, and the name of the new file10 data file to be created. Example:

```
f10filter old.f10 new.f10
```

There are comments at the top of the Fortran source program. It is recommended that the user read them.

Listed below are the two files in this subdirectory.

File number:	Sub-directory:	File name:	Purpose or contents:
207	f10filter	f10filter.f	The Fortran source of the F10FILTER program.
208	f10filter	ex1.f10	A sample data case,which can be read by the old 3DGRAPE program.

Table 16. F10FILTER Program and Data Files (Files 207 - 208)

THEORETICAL DEVELOPMENT

POISSON EQUATIONS IN PHYSICAL SPACE

The original 3DGRAPE program and the new 3DGRAPE/AL program both generate grids by iteratively solving the Poisson Equations in three-dimensions. A mapping is thus found between the computational coordinates ξ,η,ζ and the physical coordinates X,Y,Z. The equations are typically given in the computational space as

$$\xi_{xx} + \xi_{yy} + \xi_{zz} = P(\xi,\eta,\zeta) \tag{1a}$$

$$\eta_{xx} + \eta_{yy} + \eta_{zz} = Q(\xi,\eta,\zeta) \tag{1b}$$

$$\zeta_{xx} + \zeta_{yy} + \zeta_{?zz} = R(\xi,\eta,\zeta) \tag{1c}$$

However, it is natural to apply them in the physical space. It is natural to specify the grid boundary conditions by giving X,Y,Z at fixed values of ξ,η,ζ rather than to give values of ξ,η,ζ at fixed values of X,Y,Z. The transformation of Eqs. 1 to physical space proceeds as follows. Clearly, we must have

$$\xi = \xi(x,y,z) \tag{2a}$$

$$\eta = \eta(x,y,z) \tag{2b}$$

$$\zeta = \zeta(x,y,z) \tag{2c}$$

To effect this transformation we must also have

$$x = x(\xi,\eta,\zeta) \tag{3a}$$

$$y = y(\xi,\eta,\zeta) \tag{3b}$$

$$z = z(\xi, \eta, \zeta) \tag{3c}$$

Differentiating Eqs. 2 and applying the chain rule gives

$$\begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} \tag{4}$$

Likewise, differentiating Eqs. 3 and applying the chain rule gives

$$\begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} x_\xi & x_\eta & x_\zeta \\ y_\xi & y_\eta & y_\zeta \\ z_\xi & z_\eta & z_\zeta \end{vmatrix} \begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} \tag{5}$$

We designate the 3 x 3 matrix in Eq. 5 as M, assume that its inverse exists, and pre-multiply both sides of Eq. 5 by M^{-1}. This gives

$$M^{-1} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} d\xi \\ d\eta \\ d\zeta \end{vmatrix} \tag{6}$$

Substituting from Eq. 6 into Eq. 4 gives

$$M^{-1} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \begin{vmatrix} dx \\ dy \\ dz \end{vmatrix} \tag{7}$$

We know that, in general, if

$$A\vec{v} = B\vec{v} \tag{8a}$$

and if B^{-1} exists then

$$B^{-1} A \vec{v} = \vec{v} \tag{8b}$$

Therefore it must be true that

$$B^{-1} A = I \tag{8c}$$

Pre-multiplying by B gives

$$A = B \tag{8d}$$

Applying Eqs. 8 to Eq. 7 gives

$$M^{-1} = \begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} \tag{9}$$

For this to be useful, we must find M^{-1}. It is known, in general, that

$$A^{-1} = \frac{Adj(A)}{Det(A)} \tag{10}$$

Where Adj(A) is the adjoint of A and Det(A) is the determinant of A. The adjoint of A is a matrix having as each element the corresponding cofactor of A. Thus, from Eq. 9, we have

$$\begin{vmatrix} \xi_x & \xi_y & \xi_z \\ \eta_x & \eta_y & \eta_z \\ \zeta_x & \zeta_y & \zeta_z \end{vmatrix} = \begin{vmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{vmatrix} / J \tag{11}$$

where γ_{ij} is the ij-th cofactor of M and J is the determinant of M. By inspection of Eq. 11 we see that

$$\xi_x = \gamma_{11}/J \tag{12a}$$

$$\xi_y = \gamma_{12}/J \tag{12b}$$

$$\xi_z = \gamma_{13}/J \tag{12c}$$

$$\eta_x = \gamma_{21}/J \tag{12d}$$

$$\eta_y = \gamma_{22}/J \tag{12e}$$

$$\eta_z = \gamma_{23}/J \tag{12f}$$

$$\zeta_x = \gamma_{31}/J \tag{12g}$$

$$\zeta_y = \gamma_{32}/J \tag{12h}$$

$$\zeta_z = \gamma_{33}/J \tag{12i}$$

Completion of the derivation of the transformed Poisson equations requires further differentiating the metrics in Eqs. 12, substituting them into Eqs. 1, and collecting terms. This process is simple calculus, but very lengthy and beyond the scope of this TM. The result is

$$\begin{aligned} &\alpha_{11}\vec{r}_{\xi\xi} + \alpha_{22}\vec{r}_{\eta\eta} + \alpha_{33}\vec{r}_{\zeta\zeta} \\ &+ 2\,(\,\alpha_{12}\vec{r}_{\xi\eta} + \alpha_{13}\vec{r}_{\xi\zeta} + \alpha_{23}\vec{r}_{\eta\zeta}) \\ &= -J^2\,(P\vec{r}_{\xi} + Q\vec{r}_{\eta} + R\vec{r}_{\zeta}) \end{aligned} \tag{13a}$$

where:

$$\vec{r} = \begin{vmatrix} \overrightarrow{x} \\ y \\ z \end{vmatrix} \tag{13b}$$

and

$$\alpha_{ij} = \sum_{m=1}^{3} \gamma_{mi}\gamma_{mj} \tag{13c}$$

IMPROVED STEGER & SORENSON RHS TERMS

The distribution of points in the grid results primarily from the influence of the Right-Hand Side (RHS) terms, or forcing functions. We are free to choose them as we please. In both new and old programs they are:

$$\begin{aligned} P(\xi,\eta,\zeta) = {} & P_1(\eta,\zeta)e^{-a\xi} + P_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)} \\ & + P_3(\xi,\zeta)e^{-a\eta} + P_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \\ & + P_5(\xi,\eta)e^{-a\zeta} + P_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)} \end{aligned} \tag{14a}$$

$$\begin{aligned} Q(\xi,\eta,\zeta) = {} & Q_1(\eta,\zeta)e^{-a\xi} + Q_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)} \\ & + Q_3(\xi,\zeta)e^{-a\eta} + Q_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \\ & + Q_5(\xi,\eta)e^{-a\zeta} + Q_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)} \end{aligned} \tag{14b}$$

$$\begin{aligned} R(\xi,\eta,\zeta) = {} & R_1(\eta,\zeta)e^{-a\xi} + R_2(\eta,\zeta)e^{-a(\xi_{max}-\xi)} \\ & + R_3(\xi,\zeta)e^{-a\eta} + R_4(\xi,\zeta)e^{-a(\eta_{max}-\eta)} \\ & + R_5(\xi,\eta)e^{-a\zeta} + R_6(\xi,\eta)e^{-a(\zeta_{max}-\zeta)} \end{aligned} \tag{14c}$$

Clearly, these RHS terms P,Q,R are simply superpositions of other terms P_n,Q_n,R_n for $1 \le n \le 6$, multiplied by exponentials which are at their maximum value, one, at the boundary surfaces and which decay with distance into the interior of the block. The positive constant "a" in Eqs. 14 is set by the user, and determines the rate of exponential decay in the size and influence of the RHS terms.

A nomenclature for the face numbers has been introduced. It is seen in Table 7. By examining that nomenclature we see that at each of the boundaries the terms in P,Q,R having their subscripts equal to the face number are non-zero, and the other terms in P,Q,R approach zero due to the behavior of their exponential factors. At face 3, for example, Eqs. 14 reduce to:

$$P(\xi,\eta,\zeta) = P_3(\xi,\zeta) \tag{15a}$$

$$Q(\xi,\eta,\zeta) = Q_3(\xi,\zeta) \tag{15b}$$

$$R(\xi,\eta,\zeta) = R_3(\xi,\zeta) \tag{15c}$$

So then we can find the terms P_n,Q_n,R_n at face n by considering each face in turn. At each point on each face we:

- Assume that the Poisson Equations, Eqs. 13, are satisfied.
- Find values for all first and second partial derivatives required by Eqs. 13.
- Eqs. 13 reduce to a 3 x 3 set of linear equations in the three unknowns P_n,Q_n,R_n. Solve them.

Having found all the P_n,Q_n,R_n, for $1 \le n \le 6$, we can calculate P,Q,R at all points in the grid from Eqs. 14.

However, finding values for all first and second partial derivatives at each face is not trivial. To further illustrate this we must restrict our attention to a particular face. We choose face 3 to illustrate. On face 3 the derivatives $\vec{r}_\xi$, $\vec{r}_\zeta$, $\vec{r}_{\xi\zeta}$, $\vec{r}_{\xi\xi}$, and $\vec{r}_{\zeta\zeta}$ can be found by differencing known boundary face points. The derivatives $\vec{r}_{\eta\eta}$ are found by differencing the grid solution at the current time step, as described on page 78 of Ref. 2. If we could find derivatives $\vec{r}_\eta$ we could then difference them to find derivatives $\vec{r}_{\xi\eta}$ and $\vec{r}_{\eta\zeta}$.

We find derivatives $\vec{r}_\eta$ by adding additional equations which embody the user's requirements on cell height and skewness. In the old 3DGRAPE method we added the three equations

$$\vec{r}_\xi \cdot \vec{r}_\eta = 0 \tag{16a}$$

$$\vec{r}_\eta \cdot \vec{r}_\zeta = 0 \tag{16b}$$

$$\vec{r}_\eta \cdot \vec{r}_\eta = S^2 \tag{16c}$$

As seen in Table 7, ξ and ζ vary over face 3, and η varies along lines intersecting the face. Thus Eqs. 16a and 16b require orthogonality between the lines intersecting the face and the coordinate lines running over the face. Eq. 16c requires that the cell height on the surface be the positive constant S.

It is at this point that the old 3DGRAPE method and the new 3DGRAPE/AL method differ. In the new method we realize that when making grids about real-world configurations, with singularities and slope discontinuities, it is sometimes necessary to have grid cells which are skewed in a specified way. Lacking this ability, an inconsistency can develop which can either cause the elliptic solver to not converge, or result in an unsuitable grid. And so Eqs. 16 are replaced by

$$\vec{r}_\xi \cdot \vec{r}_\eta = |\vec{r}_\xi| \cdot |\vec{r}_\eta| \cos\theta_1 \tag{17a}$$

$$\vec{r}_\eta \cdot \vec{r}_\zeta = |\vec{r}_\eta| \cdot |\vec{r}_\zeta| \cos\theta_2 \tag{17b}$$

$$\vec{r}_\eta \cdot \vec{r}_\eta = S^2 \tag{17c}$$

where θ_1 is the angle between the coordinate line intersecting face 3 and the line of varying ξ on face 3, and θ_2 is the angle between the coordinate line intersecting face 3 and the line of varying ζ on face 3. For θ_1 and θ_2 equal to 90°, Eqs. 17 reduce to Eqs. 16.

We now proceed to solve Eqs. 17 for $\bar{r}_\eta$. Expanding, we have

$$x_\xi x_\eta + y_\xi y_\eta + z_\xi z_\eta = c_1 \tag{18a}$$

$$x_\zeta x_\eta + y_\zeta y_\eta + z_\zeta z_\eta = c_2 \tag{18b}$$

$$x_\eta^2 + y_\eta^2 + z_\eta^2 = S^2 \tag{18c}$$

where

$$c_1 = |\vec{r}_\xi| \, S \cos\theta_1$$

$$c_2 = |\vec{r}_\zeta| \, S \cos\theta_2$$

C_1 and c_2 are constants because θ_1, θ_2, S, and the points on face 3 are user-defined inputs. Equations 18 are three equations in the three unknowns x_η, y_η, z_η which are the elements of $\bar{r}_\eta$. But because Eq. 18c is quadratic, solving this set of equations is not straightforward. We will make an assumption about one of the unknowns and solve, make that assumption about another of the unknowns and solve, and then make that assumption about the last of the unknowns and solve. We will then select the answer which is "best."

The first assumption we make is that x_η is a constant. Terms involving x_η in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$y_\eta = x_\eta \gamma_{22} / \gamma_{12} + k_1 \tag{19a}$$

$$z_\eta = x_\eta \gamma_{32} / \gamma_{12} + k_2 \tag{19b}$$

where

$$k_1 = \frac{c_1 z_\zeta - c_2 z_\xi}{-\gamma_{12}}$$

$$k_2 = \frac{c_2 y_\xi - c_1 y_\zeta}{-\gamma_{12}}$$

K1 and k2 are constants. Then from Eq. 18c

$$x_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{20}$$

where

$$a = 1 + (\gamma_{22}/\gamma_{12})^2 + (\gamma_{32}/\gamma_{12})^2$$

$$b = \frac{2}{\gamma_{12}}(k_1\gamma_{12} + k_2\gamma_{32})$$

$$c = k_1^2 + k_2^2 - S^2$$

The second assumption is that y_η is a constant. Terms involving y_η in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$x_\eta = y_\eta\gamma_{12} / \gamma_{22} + k_1 \tag{21a}$$

$$z_\eta = y_\eta\gamma_{32} / \gamma_{22} + k_2 \tag{21b}$$

where

$$k_1 = \frac{c_1 z_\zeta - c_2 z_\xi}{\gamma_{22}}$$

$$k_2 = \frac{c_2 x_\xi - c_1 x_\zeta}{\gamma_{22}}$$

Then from Eq. 18c

$$y_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{22}$$

where

$$a = 1 + (\gamma_{12}/\gamma_{22})^2 + (\gamma_{32}/\gamma_{22})^2$$

$$b = \frac{2}{\gamma_{22}}(k_1\gamma_{12} + k_2\gamma_{32})$$

and c is the same as above, in Eq. 20.

The third assumption is that z_η is a constant. Terms involving z_η in Eqs. 18a and 18b are brought to the right side of the equations, and then the equations are solved, yielding

$$x_\eta = z_\eta \gamma_{12} / \gamma_{32} + k_1 \tag{23a}$$

$$y_\eta = z_\eta \gamma_{22} / \gamma_{32} + k_2 \tag{23b}$$

where

$$k_1 = \frac{c_1 y_\zeta - c_2 y_\xi}{-\gamma_{32}}$$

$$k_2 = \frac{c_2 x_\xi - c_1 x_\zeta}{-\gamma_{32}}$$

Then from Eq. 17c

$$z_\eta = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{24}$$

where

$$a = 1 + (\gamma_{12}/\gamma_{32})^2 + (\gamma_{22}/\gamma_{32})^2$$

$$b = \frac{2}{\gamma_{32}}(k_1 \gamma_{12} + k_2 \gamma_{22})$$

and c is the same as above, in Eq. 20.

In general, none of these three assumptions is strictly correct. However, it usually turns out that at least one of them is close enough to correct for this method to generate suitable grids. It was said that we would choose whichever of these three solutions was "best." However, Eqs. 20, 22, and 24 each include an ambiguous sign from a square-root operation. Therefore, we actually have six solutions to choose from. Using each of the six solutions we compute the Jacobian. If the coordinates in the block are right-handed (with the "handedness" being a user-defined input) we choose the solution which yields the largest positive Jacobian. If the coordinates in the block are left-handed we choose the solution which yields the largest negative Jacobian. The logic behind choosing based upon the Jacobian is that Jacobians, as defined above, having large absolute values seem to be present in grids which are more orthogonal, and, conversely, Jacobians having small absolute values seem to be present in grids which are highly skewed. Thus the elements of r_η are found.

The foregoing is the analysis for face 3. The analysis for face 4 appears nearly identical, differing only in some of the difference formulas. The analyses for faces 1, 2, 5, and 6 follow in a straightforward manner from the foregoing example.

This formulation for the S&S RHS terms requires a lot of computation but most of it is done only once, at the start of the iteration schedule. It was said, above, that having all values for the derivatives at the face those derivatives are substituted into Eqs. 13,

yielding a 3 x 3 set of linear equations in the three unknowns P_n,Q_n,R_n. Their solution shows P_n,Q_n,R_n to be linear functions of the second derivatives $\vec{r}_{\eta\eta}$ which are found by differencing at each time step. The coefficients in those linear functions are fixed for all computational time. Therefore, the only computation necessary to find the RHS terms in each iteration is to re-evaluate $\vec{r}_{\eta\eta}$, re-compute the linear functions using the fixed coefficients to get P_n,Q_n,R_n at each face, and then use Eqs. 14 to re-compute the P,Q,R at every point in the grid.

The effectiveness of this method is seen in Figure 4. When wrapping a grid around a sharp edge it is necessary to cause the lines intersecting the surface near the edge to bend toward the edge for best results. The ultimate example of wrapping a grid around a sharp edge is to wrap it around the edge of a flat plate. Figure 4 shows a wing with a zero-thickness extension in the spanwise direction, and a C-H type grid around it. Thus, it is necessary to wrap a C-type grid around the leading-edge of that wing and its flat-plate extension. This would not have been possible with the old type RHS terms.

THOMAS & MIDDLECOFF CLUSTERING TERMS

When making grids in regions where all six faces of the computational cube are fixed it is sometimes advantageous to use clustering functions where the spacing normal to a face is determined by the spacing on the side walls. The Thomas and Middlecoff clustering terms, described in Ref. 6, are included here for that purpose. However, the Thomas and Middlecoff clustering terms

$$P = \Phi(\nabla\xi \cdot \nabla\xi) \tag{25a}$$

$$Q = \Psi(\nabla\eta \cdot \nabla\eta) \tag{25b}$$

$$R = \Omega(\nabla\zeta \cdot \nabla\zeta) \tag{25c}$$

where

$$\Phi = \frac{\vec{r}_\xi \cdot \vec{r}_{\xi\xi}}{\vec{r}_\xi \cdot \vec{r}_\xi} \tag{25d}$$

$$\Psi = \frac{\vec{r}_\eta \cdot \vec{r}_{\eta\eta}}{\vec{r}_\eta \cdot \vec{r}_\eta} \tag{25e}$$

$$\Omega = \frac{\vec{r}_\zeta \cdot \vec{r}_{\zeta\zeta}}{\vec{r}_\zeta \cdot \vec{r}_\zeta} \tag{25f}$$

are given in the computational space, and to be useful here they must be converted to physical space. Applying the definition of the ∇ operator, illustrated by

$$\nabla\xi = \xi_x\hat{j} + \xi_y\hat{k} + \xi_z\hat{l} \tag{26}$$

where $\hat{j}$, $\hat{k}$, and $\hat{l}$ are the unit normal vectors, and reducing, gives

$$P = \Phi(\xi_x^2 + \xi_y^2 + \xi_z^2) \tag{27a}$$

$$Q = \psi(\eta_x^2 + \eta_y^2 + \eta_z^2) \tag{27b}$$

$$R = \Omega(\zeta_x^2 + \zeta_y^2 + \zeta_z^2) \tag{27c}$$

Substituting the metrics shown in Eqs. 12 into Eqs. 27, and expanding and re-grouping, gives

$$P = \Phi\left[\left(\vec{r}_\eta \cdot \vec{r}_\eta\right)\left(\vec{r}_\zeta \cdot \vec{r}_\zeta\right) - \left(\vec{r}_\eta \cdot \vec{r}_\zeta\right)\right]/J^2 \tag{28a}$$

$$Q = \psi\left[\left(\vec{r}_\xi \cdot \vec{r}_\xi\right)\left(\vec{r}_\zeta \cdot \vec{r}_\zeta\right) - \left(\vec{r}_\xi \cdot \vec{r}_\zeta\right)\right]/J^2 \tag{28b}$$

$$R = \Omega\left[\left(\vec{r}_\xi \cdot \vec{r}_\xi\right)\left(\vec{r}_\eta \cdot \vec{r}_\eta\right) - \left(\vec{r}_\xi \cdot \vec{r}_\eta\right)\right]/J^2 \tag{28c}$$

These RHS terms generate good grids in many applications. An exception is the situation where the opposing side boundaries, from which the T&M terms are calculated, have very different clustering characteristics. In these cases instabilities in the Poisson solver can result.

It was found in the development of 3DMAGGS (Ref. 4) that S&S clustering terms tend to give the most-nearly-orthogonal grids near boundaries, while T&M clustering terms give the best clustering in the interior of the blocks. And so a blending between the two kinds of RHS terms was developed, and is included in 3DGRAPE/AL.

OPTIMUM RELAXATION PARAMETER

3DGRAPE/AL solves the 3-D Poisson equations using Point Successive Over Relaxation (PSOR). In PSOR there is a relaxation parameter, Ω, which determines the rate of convergence and stability of the method. In the old program the Ω was fixed for all computational time. That option is still available in the new code as well. However, the new code also has an algorithm to compute an optimum relaxation parameter at every point in the grid using the method of Erlich, as described in Ref. 8.

That method requires the equations being solved, here Eq. 13a, to be represented as a difference equation of the following form:

$$\begin{aligned} a_0\vec{r}_{j,k,l} + a_1\vec{r}_{j+1,k,l} + a_2\vec{r}_{j,k+1,l} + a_3\vec{r}_{j,k,l+1} \\ + \quad a_4\vec{r}_{j-1,k,l} + a_5\vec{r}_{j,k-1,l} + a_6\vec{r}_{j,k,l-1} = \vec{b}_{j,k,l} \end{aligned} \tag{29}$$

Applying standard central differences to all first and second partial derivatives in Eq. 13a, and collecting terms, we arrive at the form of Eq. 29, where

$$a_0 = -2\left(\frac{\alpha_{11}}{(\Delta\xi)^2} + \frac{\alpha_{22}}{(\Delta\eta)^2} + \frac{\alpha_{33}}{(\Delta\zeta)^2}\right) \tag{30a}$$

$$a_1 = \frac{\alpha_{11}}{(\Delta\xi)^2} + \frac{J^2 P}{2\Delta\xi} \tag{30b}$$

$$a_2 = \frac{\alpha_{22}}{(\Delta\eta)^2} + \frac{J^2 Q}{2\Delta\eta} \tag{30c}$$

$$a_3 = \frac{\alpha_{33}}{(\Delta\zeta)^2} + \frac{J^2 R}{2\Delta\zeta} \tag{30d}$$

$$a_4 = \frac{\alpha_{11}}{(\Delta\xi)^2} - \frac{J^2 P}{2\Delta\xi} \tag{30e}$$

$$a_5 = \frac{\alpha_{22}}{(\Delta\eta)^2} - \frac{J^2 Q}{2\Delta\eta} \tag{30f}$$

$$a_6 = \frac{\alpha_{33}}{(\Delta\zeta)^2} - \frac{J^2 R}{2\Delta\zeta} \tag{30g}$$

The complex eigenvalues of Eq. 29 at each point, ignoring wave numbers above 1, are

$$\mu = \mu_r + \mu_i = \frac{2}{a_0}\left(\sqrt{a_1 a_4}\cos\frac{\pi}{j_{max}+1} + \sqrt{a_2 a_5}\cos\frac{\pi}{k_{max}+1} + \sqrt{a_3 a_6}\cos\frac{\pi}{l_{max}+1}\right) \tag{31}$$

where μ_r and μ_i are the real and imaginary parts of μ, respectively. It is required that $|\mu_r| < 1$.

Continuing with Erlich's method, as formulated by Steinbrenner, Chawner, and Fouts on pages 6-6 and 6-7 of Ref. 9 (with typographical errors corrected), we let

$$A = \mu_r^2 + \mu_i^2 \tag{32a}$$

$$B = \mu_r^2 - \mu_i^2 \tag{32b}$$

$$C = A^2 - B^2 \tag{32c}$$

$$D = A^2 - B \tag{32d}$$

$$E = \sqrt{C + D^2} \tag{32e}$$

$$F = \sqrt[3]{C} \tag{32f}$$

Then

$$\overline{\omega} = \frac{(3D + E)\, F \sqrt[3]{E-D} - (3D - E)\, F \sqrt[3]{E+D} + A^2 + 3B^2 - 4A^2B}{A^2D} \tag{33}$$

and the relaxation parameter ω is

$$\omega = \begin{cases} -\left(\overline{\omega} - \sqrt{\overline{\omega}^2 + 4\overline{\omega}}\right)/2 & \text{if } D\rangle 0 \\ -\left(\overline{\omega} + \sqrt{\overline{\omega}^2 + 4\overline{\omega}}\right)/2 & \text{if } D\langle 0 \end{cases} \tag{34}$$

This method can reduce the number of iterations required to achieve convergence. The ω so computed can sometimes be a little too large, and so cause instabilities. Therefore, in the code, they are multiplied by a limiting factor. The default value of this factor is 0.7, but a value of 0.6 was found to be necessary in one of the sample cases. These ω are dependent on the grid at its current time step, and so they are typically re-calculated each time step. As can be inferred from the above, computing them requires a significant amount of computer time, so the code has an option wherein they are re-calculated every n time steps.

POSTSCRIPT FILES

The last group of files on the tape are compressed PostScript files comprising this manual. It should be possible to "uncompress" them and print them on any PostScript-compatible printer. After inserting the figure pages in the appropriate places, it should then be possible to duplicate this manual.

File number:	Sub-directory:	Filename:	Purpose or contents:
209	ps	manual.text.ps.Z	The text of this manual, in compressed PostScript form
210	ps	fig.1.ps.Z	Figure 1 in compressed PostScript form
211	ps	fig.2a.ps.Z	Figure 2a in compressed PostScript form
212	ps	fig.2b.ps.Z	Figure 2b in compressed PostScript form
213	ps	fig.3a.ps.Z	Figure 3a in compressed PostScript form
214	ps	fig.3b.ps.Z	Figure 3b in compressed PostScript form
215	ps	fig.4a.ps.Z	Figure 4a in compressed PostScript form
216	ps	fig.4b.ps.Z	Figure 4b in compressed PostScript form
217	ps	fig.4c.ps.Z	Figure 4c in compressed PostScript form
218	ps	fig.5a.ps.Z	Figure 5a in compressed PostScript form
219	ps	fig.5b.ps.Z	Figure 5b in compressed PostScript form
220	ps	fig.6a.ps.Z	Figure 6a in compressed PostScript form

221	ps	fig.6b.ps.Z	Figure 6b in compressed PostScript form
222	ps	fig.7a.ps.Z	Figure 7a in compressed PostScript form
223	ps	fig.7b.ps.Z	Figure 7b in compressed PostScript form
224	ps	fig.7c.ps.Z	Figure 7c in compressed PostScript form
225	ps	fig.7d.ps.Z	Figure 7d in compressed PostScript form
226	ps	fig.7e.ps.Z	Figure 7e in compressed PostScript form
227	ps	fig.8a.ps.Z	Figure 8a in compressed PostScript form
228	ps	fig.8b.ps.Z	Figure 8b in compressed PostScript form
229	ps	fig.9a.ps.Z	Figure 9a in compressed PostScript form
230	ps	fig.9b.ps.Z	Figure 9b in compressed PostScript form

Table 17. PostScript Files Comprising This Manual (Files 209 - 230)

THE END

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1995	Reference Publication

4. TITLE AND SUBTITLE

3DGRAPE/AL Users' Manual

5. FUNDING NUMBERS

505-59-53

6. AUTHOR(S)

Reese L. Sorenson and Stephen J. Alter*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ames Research Center
Moffett Field, CA 94035-1000

8. PERFORMING ORGANIZATION REPORT NUMBER

A-950088

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA RP-1377

11. SUPPLEMENTARY NOTES

Point of Contact: Reese L. Sorenson, Ames Research Center, MS T27B-2,Moffett Field, CA 94035-1000; (415) 604-4471

*Lockheed Martin Engineering & Sciences, Hampton, VA 23666

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category – 61

Available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document is a users' manual for a new three-dimensional structured multiple-block volume grid generator called 3DGRAPE/AL. It is a significantly improved version of the previously-released and widely-distributed programs 3DGRAPE and 3DMAGGS. It generates volume grids by iteratively solving the Poisson Equations in three-dimensions. The right-hand-side terms are designed so that user-specified grid cell heights and user-specified grid cell skewness near boundary surfaces result automatically, with little user intervention. The code is written in Fortran-77, and can be installed with or without a simple graphical user interface which allows the user to watch as the grid is generated.

An introduction describing the improvements over the antecedent 3DGRAPE code is presented first. Then follows a chapter on the basic grid generator program itself, and comments on installing it. The input is then described in detail. After that is a description of the Graphical User Interface. Five example cases are shown next, with plots of the results. Following that is a chapter on two input filters which allow the use of input data generated elsewhere. Last is a treatment of the theory embodied in the code.

14. SUBJECT TERMS	15. NUMBER OF PAGES
GRAPE, Grid generation, CFD	125
	16. PRICE CODE: A06

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		